Creating a Digital Ecosystem: Service-Oriented Architectures with Distributed Evolutionary Computing

Gerard Briscoe
Intelligent Systems and Networks Group
Department of Electrical and Electronic Engineering
Imperial College London

arXiv:0712.4159v3 [cs.NE] 5 Oct 2009

Abstract

We start with a discussion of the relevant literature, including Nature Inspired Comuting as a framework in which to understand this work, and the process of biomimicry to be used in mimicking the necessary biological processes to create Digital Ecosystems. We then consider the relevant theoretical ecology in creating the digital counterpart of a biological ecosystem, including the topological structure of ecosystems, and evolutionary processes within distributed environments. This leads to a discussion of the relevant fields from computer science for the creation of Digital Ecosystems, including evolutionary computing, Multi-Agent Systems, and Service-Oriented Architectures. We then define Ecosystem-Oriented Architectures for the creation of Digital Ecosystems, imbibed with the properties of self-organisation and scalability from biological ecosystems, including a novel form of distributed evolutionary computing.

1 Introduction

Is mimicking ecosystems the future of information systems ?

A key challenge in modern computing is to develop systems that address complex, dynamic problems in a scalable and efficient way, because the increasing complexity of software makes designing and maintaining efficient and flexible systems a growing challenge [84, 123, 76]. What with the ever expanding number of services being offered online from Application Programming Interfaces (APIs) being made public, there is an ever growing number of computational units available to be combined in the creation of applications. However, this is currently a task done manually by programmers, and it has been argued that current software development techniques have hit a *complexity wall* [72], which can only be overcome by automating the search for new algorithms. There are several existing efforts aimed at achieving this automated service composition [80, 92, 82, 105], the most prevalent of which is Service-Oriented Architectures and its associated standards and technologies [24, 130].

Alternatively, nature has been in the research business for 3.8 billion years and in that time has accumulated close to 30 million *well-adjusted* solutions to a plethora of design challenges that humankind struggles to address with mixed results [13]. Biomimicry is a discipline that seeks solutions by emulating nature's designs and processes, and there is considerable opportunity to learn elegant solutions for human-made problems [13]. Biological ecosystems are thought

to be robust, scalable architectures that can automatically solve complex, dynamic problems, possessing several properties that may be useful in automated systems. These properties include self-organisation, self-management, scalability, the ability to provide complex solutions, and automated composition of these complex solutions [67].

Therefore, an approach to the aforementioned challenge would be to develop Digital Ecosystems, artificial systems that aim to harness the dynamics that underlie the complex and diverse adaptations of living organisms in biological ecosystems. While evolution may be well understood in computer science under the auspices of *evolutionary computing* [32], ecological models are not. The possible connections between Digital Ecosystems and their biological counterparts are yet to be closely examined, so potential exists to create an Ecosystem-Oriented Architecture with the essential elements of biological ecosystems, where the word *ecosystem* is more than just a *metaphor*. We propose that an ecosystem inspired approach, would be more effective at greater scales than traditionally inspired approaches, because it would be built upon the scalable and self-organising properties of biological ecosystems [67].

2 Background Theory

Our focus is in creating the digital counterpart of biological ecosystems. However, the term *digital ecosystem* has been used to describe a variety of concepts, which it now makes sense to review. Some of these refer to the existing networking infrastructure of the internet [29, 33, 138], while several companies offer a *digital ecosystem* service or solution, which involves enabling customers to use existing e-business solutions [12, 60, 128]. The term is also being increasingly linked, yet undefined, to the future developments of Information and Communications Technology (ICT) adoption for e-business and e-commerce, to create so called *business ecosystems* [52, 89, 96]. However, perhaps the most frequent references to *digital ecosystems* arise in Artificial Life research, where they are created primarily to investigate aspects of biological and other complex systems [120, 40, 21].

The extent to which these disparate systems resemble biological ecosystems varies, and frequently the word *ecosystem* is merely used for branding purposes without any inherent ecological properties. We consider Digital Ecosystems to be software systems that exploit the properties of biological ecosystems, and suggest that several key features of biological ecosystems have not been fully explored in existing *digital ecosystems*. So, we will now discuss how mimicking these features can create Digital Ecosystems, which are robust, scalable, and self-organising.

2.1 Nature-Inspired Computing

Biomimicry (bios, meaning life, and mimesis, meaning to imitate) is the science that studies nature, its models, systems, processes, and elements, and then imitates or takes creative inspiration from them for the study and design of engineering systems and modern technology [13]. This concept is far from new, with humans having long been inspired by the animals and plants of the natural world; Leonardo Da Vinci himself once said, *Those who are inspired by a model other than Nature, a mistress above all masters, are labouring in vain* [15]. Albeit overstating the point, it reminds us that the transfer of technology between life-forms and synthetic constructs is desirable because evolutionary pressures typically force living organisms to become highly optimised and efficient. A classical example is the development of dirt and water repellent paint from the observation that the surface of the lotus flower plant is practically un-sticky for anything, commonly known as the *lotus effect* [8]. However, biomimicry, when done well, is not slavish imitation; it is inspiration using the principles which nature has demonstrated to be successful design strategies. In the early days of mechanised flight the best designs were not

the ornithopters, which most completely imitated birds, but the fixed-wing craft that used the principle of airfoil cross-section in their wings [2].

Biomimicry in computer science is called Nature Inspired Computing (NIC) or Natural Computation, and the benefits of natural computation technologies often mimic those found in real natural systems, and include flexibility, adaptability, robustness, and decentralised control [26]. The increasing demands upon current computer systems, along with technological changes, create a need for more flexible and adaptable systems. The desire to achieve this has led many computing researchers to look to natural systems for inspiration in the design of computer software and hardware, as natural systems provide many examples of the type of versatile system required [26]. Their sources of inspiration come from many aspects of natural systems; evolution, ecology, development, cell and molecular phenomena, behaviour, cognition, and other areas [77]. The use of nature inspired techniques often results in the design of novel computing systems with applicability in many different areas [77]. NIC itself can be divided into three main branches [26]:

- Biologically-Inspired Computing (BIC): This makes use of nature as inspiration for the development of problem solving techniques. The main idea of this branch is to develop computational tools (algorithms) by taking inspiration from nature for the solution of complex problems.

- The simulation and emulation of nature by computational means: This is basically a synthetic process aimed at creating patterns, forms, behaviours, and organisms that resemble *life-as-we-know-it*. Its products can be used to mimic various natural phenomena, thus increasing our understanding of nature and insights about computer models.

- Computing with natural materials: This corresponds to the use of natural materials to perform computation, to substitute or supplement the current silicon-based computers.

All branches share the common characteristic of human-designed computing inspired by nature, the metaphorical use of concepts, principles, and mechanisms underlying natural systems. Thus, evolutionary algorithms use the concepts of mutation, recombination, and natural selection from biology; neural networks are inspired by the highly interconnected neural structures in the brain and the nervous system; molecular computing is based on paradigms from molecular biology; and quantum computing based on quantum physics exploits quantum parallelism [26]. There are however, important methodological differences between various sub-areas of natural computing. For example, evolutionary algorithms and algorithms based on neural networks are presently implemented on conventional computers. However, molecular computing also aims at alternatives for silicon hardware by implementing algorithms in biological hardware, using DNA molecules and enzymes. Also, quantum computing aims at non-traditional hardware that can make use of quantum effects [26].

We are concerned with BIC, which relies heavily on the fields of biology, computer science, and mathematics. Briefly put, it is the study of nature to improve the usage of computers [37], and should not to be confused with *computational biology* [132], which is an interdisciplinary field that applies the techniques of computer science, applied mathematics, and statistics to address problems inspired by biology. BIC has produced Neural Networks, swarm intelligence and evolutionary computing [37]. Introducing BIC, one comes quickly to its applications, partly because this is the essence of the approach, and partly because biomimicry as a process tends to be ad-hoc and un-formalised [48]. It generally involves an engineer or scientist observing or being aware of an area of biological study, which seems applicable to a technology or research problem they are currently tackling, or which inspires the creation of a new technology [26]. However, there are some common steps in this process, which starts with identifying some

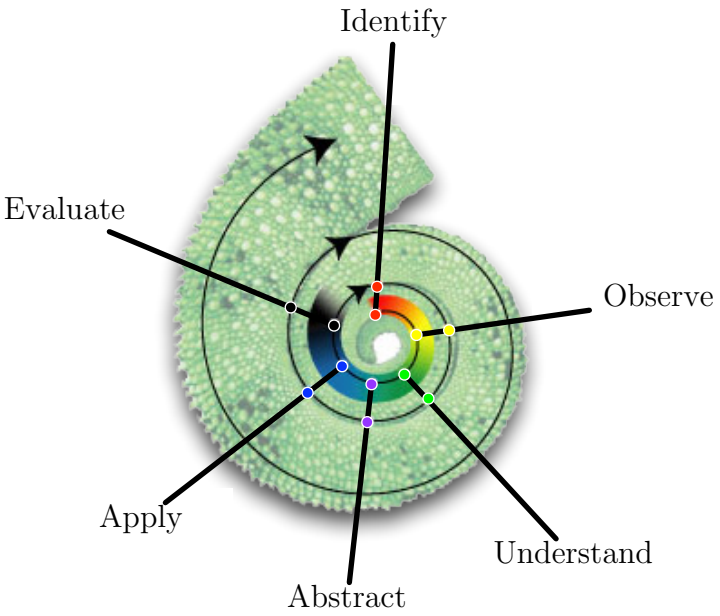

Figure 1: *Biomimicry Design Spiral (modified from [48]): The process of biomimicry starts with identifying some behaviour from a biological system, which would appear to be useful. Followed by observation to understand the mechanisms or principles by which it operates, and allowing for an abstract understanding of the behaviour. This can then be mimicked in a non-biological system and its performance and effectiveness evaluated [48].*

behaviour from a biological system, which would appear to be useful. Followed by observation to understand the mechanisms or principles by which it operates, and allowing for an abstract understanding of the behaviour. This can then be mimicked in a non-biological system and its performance and effectiveness evaluated [48]. This process is summarised Figure 1.

2.2 Biology of Digital Ecosystems

Natural science is the study of the universe via the rules or laws of natural order, and the term *natural science* is also used to differentiate those fields using scientific method in the study of nature, in contrast with the social sciences which apply the scientific method to culture and human behaviour: economics, psychology, political economy, anthropology, etc [50]. The fields of natural science are diverse, ranging from particle physics to astronomy [111], and while not all these fields of study will provide paradigms for Digital Ecosystems, the further one wishes to take the analogy of the word *ecosystem*, the more one has to consider the relevance of the fields of natural science, particularly the biological sciences.

A primary motivation for our research in Digital Ecosystems is the desire to exploit the self-organising properties of biological ecosystems. Ecosystems are thought to be robust, scalable architectures that can automatically solve complex, dynamic problems [67]. However, the biological processes that contribute to these properties have not been made explicit in Digital Ecosystems research. Here, we discuss how biological properties contribute to the self-organising features of biological ecosystems, including population dynamics, evolution, a complex dynamic environment, and spatial distributions for generating local interactions [126]. The potential for exploiting these properties in artificial systems is then considered. We suggest that several key features of biological ecosystems have not been fully explored in existing digital ecosystems, and discuss how mimicking these features may assist in developing robust, scalable self-organising architectures.

Genetic algorithms are a form of evolutionary computing, and like all forms uses natural selection to evolve solutions [39]; started with a set of possible solutions chosen arbitrarily,

then selection, replication, recombination, and mutation are applied iteratively. Selection is based on conforming to a fitness function which is determined by a specific problem of interest, and so over time better solutions to the problem can thus evolve [39]. As Digital Ecosystems will likely solve problems by evolving solutions, they will probably incorporate some form of evolutionary computing. However, we suggest that Digital Ecosystems should also incorporate additional features, providing it with a closer resemblance to biological ecosystems. Including features such as complex dynamic fitness functions, a distributed or network environment, and self-organisation arising from interactions among organisms and their environment, which we will discuss later.

Arguably the most fundamental differences between biological and digital ecosystems lie in the motivation and approach of their respective researchers. Biological ecosystems are ubiquitous natural phenomena whose maintenance is crucial to our survival, developing through the process of *ecological succession* [9]. In contrast, Digital Ecosystems will be defined here as a technology engineered to serve specific human purposes, developing to solve dynamic problems in parallel with high efficiency.

2.2.1 Biological Ecosystems

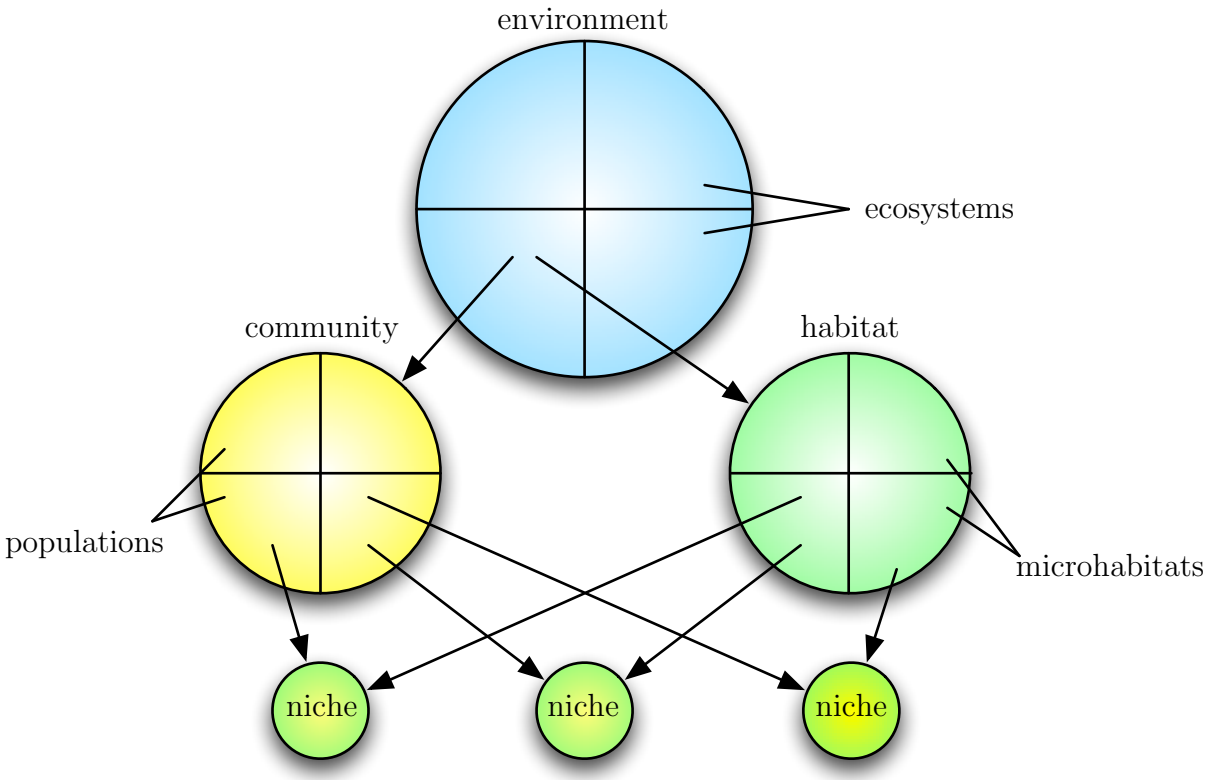

Figure 2: *Ecosystem Structure (redrawn from [107]): A stable, self-perpetuating system made up of one or more communities of organisms, consisting of species in their habitats, with their populations existing in their respective micro-habitats [9]. A community is a naturally occurring group of populations from different species that live together, and interact as a self-contained unit in the same habitat. A habitat is a distinct part of the environment [9].*

An ecosystem is a natural unit made up of living (biotic) and non-living (abiotic) components, from whose interactions emerge a stable, self-perpetuating system. It is made up of one or more communities of organisms, consisting of species in their habitats, with their populations existing in their respective micro-habitats [9]. A community is a naturally occurring group of populations from different species that live together, and interact as a self-contained unit in the same habitat. A habitat is a distinct part of the environment [9], for example, a stream. Individual organisms migrate through the ecosystem into different habitats competing

with other organisms for limited resources, with a population being the aggregate number of the individuals, of a particular species, inhabiting a specific habitat or micro-habitat [9]. A micro-habitat is a subdivision of a habitat that possesses its own unique properties, such as a micro-climate [66]. Evolution occurs to all living components of an ecosystem, with the evolutionary pressures varying from one population to the next depending on the environment that is the population's habitat. A population, in its micro-habitat, comes to occupy a niche, which is the functional relationship of a population to the environment that it occupies. A niche results in the highly specialised adaptation of a population to its micro-habitat [66].

2.2.2 Fitness Landscapes and Agents

As described above, an ecosystem comprises both an environment and a set of interacting, reproducing entities (or agents) in that environment; with the environment acting as a set of physical and chemical constraints on reproduction and survival [9]. These constraints can be considered in abstract using the metaphor of the fitness landscape, in which individuals are represented as solutions to the problem of survival and reproduction [137]. All possible solutions are distributed in a space whose dimensions are the possible properties of individuals. An additional dimension, height, indicates the relative fitness (in terms of survival and reproduction) of each solution. The fitness landscape is envisaged as a rugged, multidimensional landscape of hills, mountains, and valleys, because individuals with certain sets of properties are *fitter* than others [137], as visualised in Figure 3.

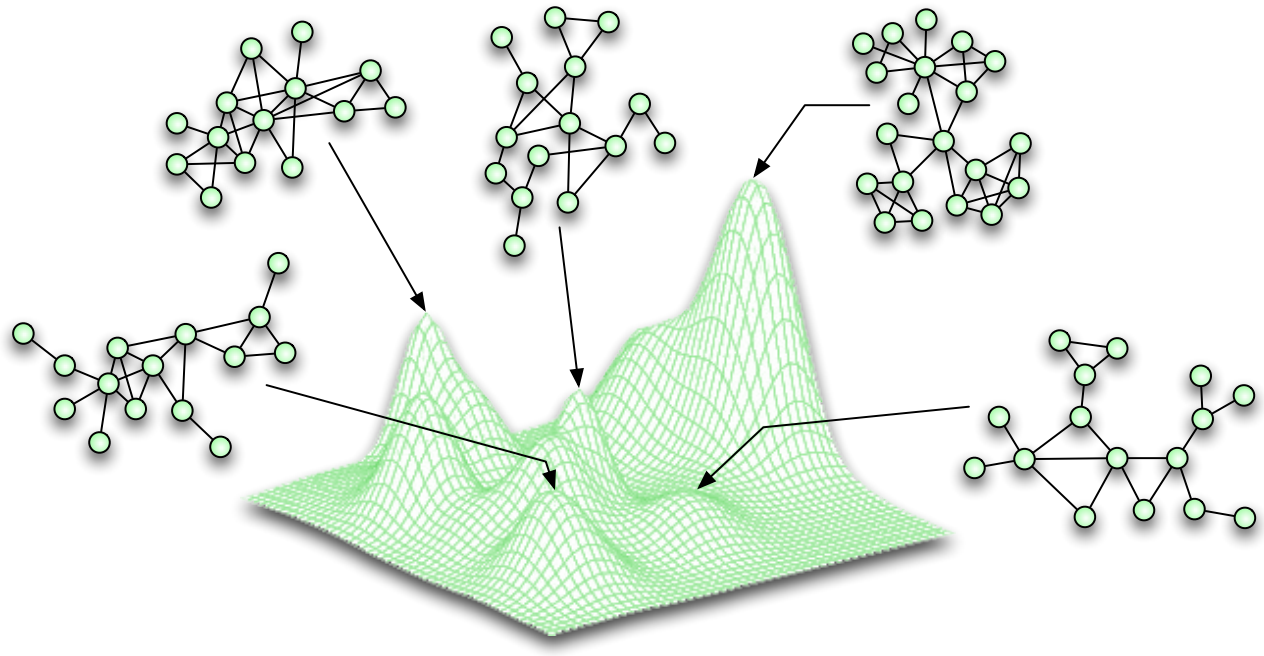

Figure 3: *Fitness Landscape (modified from [127]): We can represent software development as a walk through the landscape, towards the peaks which correspond to the optimal applications. Each point represents a unique combination of software services, and the roughness of the landscape indicates how difficult it is to reach an optimal software design [127]. In this example, there is a global optimum, and several lower local optima.*

In biological ecosystems, fitness landscapes are virtually impossible to identify. This is both because there are large numbers of possible traits that can influence individual fitness, and because the environment changes over time and space [9]. In contrast, within a digital environment, it is normally possible to specify explicitly the constraints that act on individuals in order to evolve solutions that perform better within these constraints. Within genetic algorithms,

exact specification of a fitness landscape or function is common practice [39]. However, within a Digital Ecosystem the ideal constraints are those that allow solution populations to evolve to meet user needs with maximum efficiency. User needs will change from place to place and time to time. In this sense the fitness landscape of a Digital Ecosystem is complex and dynamic, and more like that of a biological ecosystem than like that of a traditional genetic algorithm [87, 39]. The designer of a Digital Ecosystem therefore faces a double challenge: firstly, to specify rules that govern the shape of the fitness function/landscape in a way that meaningfully maps landscape dynamics to user requests, and secondly, to evolve within this space, solution populations that are diverse enough to solve disparate problems, complex enough to meet user needs, and efficient enough to be preferable to those generated by other means.

The agents within a Digital Ecosystem will need to be like biological individuals in the sense that they reproduce, vary, interact, move, and die [9]. Each of these properties contributes to the dynamics of the ecosystem. However, the way in which these individual properties are encoded may vary substantially depending on the intended purpose of the system [20].

2.2.3 Networks and Spatial Dynamics

A key factor in the maintenance of diversity in biological ecosystems is spatial interactions, and several modelling systems have been used to represent these spatial interactions. Including metapopulations[1], diffusion models, cellular automata and agent-based models (termed individual-based models in ecology) [42]. The broad predictions of these diverse models are in good agreement. At local scales, spatial interactions favor relatively abundant species disproportionately. However, at a wider scale, this effect can preserve diversity, because different species will be locally abundant in different places. The result is that even in homogeneous environments, population distributions tend to form discrete; long-lasting patches that can resist an invasion by superior competitors [42]. Population distributions can also be influenced by environmental variations such as barriers, gradients, and patches. The possible behaviour of spatially distributed ecosystems is so diverse that scenario-specific modelling is necessary to understand any real system [45]. Nonetheless, certain robust patterns are observed. These include the relative abundance of species, which consistently follows a roughly log-normal relationship [10], and the relationship between geographic area and the number of species present, which follows a power law [117]. The reasons for these patterns are disputed, because they can be generated by both spatial extensions of simple Lotka-Volterra competition models [51], and more complex ecosystem models [119].

Landscape connectivity plays an important part in ecosystems. When the density of habitats within an environment falls below a critical threshold, widespread species may fragment into isolated populations. Fragmentation can have several consequences. Within populations, these effects include loss of genetic diversity and detrimental inbreeding [41]. At a broader scale, isolated populations may diverge genetically, leading to speciation, as shown in Figure 4.

From an information theory perspective, this phase change in landscape connectivity can mediate global and local search strategies [44]. In a well-connected landscape, selection favors the globally superior, and pursuit of different evolutionary paths is discouraged, potentially leading to premature convergence. When the landscape is fragmented, populations may diverge, solving the same problems in different ways. Recently, it has been suggested that the evolution of complexity in nature involves repeated landscape phase changes, allowing selection to alternate between local and global search [43].

In a digital context, we can have spatial interactions by using a distributed system that consists of a set of interconnected locations, with agents that can migrate between these con-

[1] A metapopulation is a collection of relatively isolated, spatially distributed, local populations bound together by occasional dispersal between populations. [69, 46, 47]

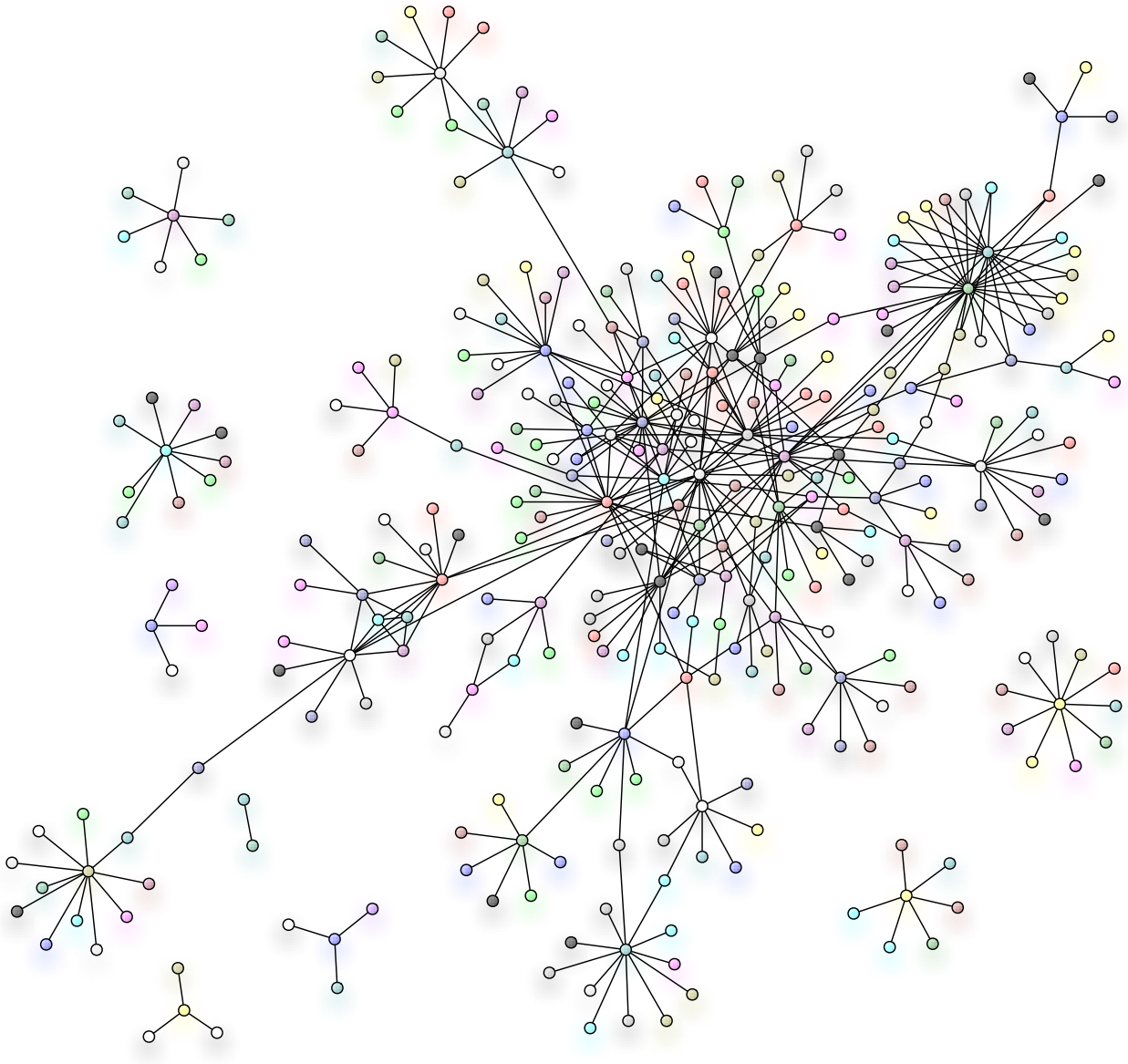

Figure 4: *Abstract View of An Ecosystem: Showing different populations (by the different colours) in different spatial areas, and their connection to one another by the lines. Included are communities of populations that have become geographically separated and so are not connected to the main network of the ecosystem, and which could potentially give rise to allopatric (geographic) speciation [66].*

nected locations. In such systems the spatial dynamics are relatively simple compared with those seen in real ecosystems, which incorporate barriers, gradients, and patchy environments at multiple scales in continuous space [9]. Nevertheless, depending on how the connections between locations are organised, such Digital Ecosystems might have dynamics closely parallel to spatially explicit models, diffusion models, or metapopulations [45]. We will discuss later the use of a dynamic non-geometric spatial network, and the reasons for using this approach.

2.2.4 Selection and Self-Organisation

The major hypothetical advantage of Digital Ecosystems over other complex organisational models is their potential for dynamic adaptive self-organisation. However, for the solutions evolving in Digital Ecosystems to be useful, they must not only be efficient in a computational

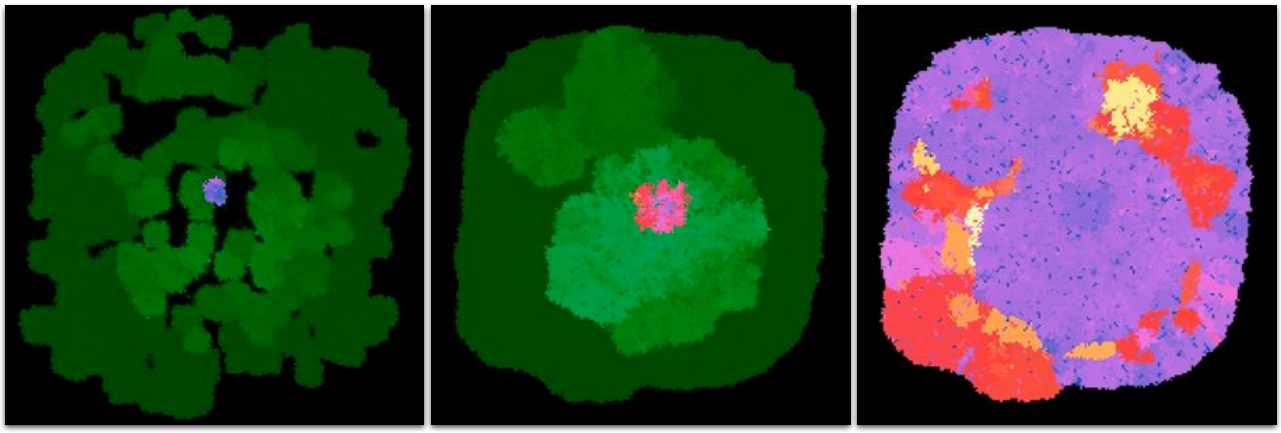

Figure 5: *Evolving Population of Digital Organisms: A virtual petri dish at three successive time-steps, showing the self-organisation of the population undergoing selection. The colour shows the genetic variability of the digital organisms. Over time the fitter (purple) organisms come to dominate the population, reproducing more and essentially replacing the weaker organisms of the population [100].*

sense, but they must also solve purposeful problems. That is, the fitness of agents must translate in some sense to real-world usefulness as demanded by the users [30].

Constructing a useful Digital Ecosystem therefore requires a balance between freedom of the system to self-organise, and constraint of the system to generate useful solutions. These factors must be balanced because the more the system's behaviour is dictated by its internal dynamics, the less it may respond to fitness criteria imposed by the users. At one extreme, when system dynamics are mainly internal, agents may evolve that are good at survival and reproduction within the digital environment, but useless in the real world [30]. At the other extreme, where the users' fitness criteria overwhelmingly dictates function, we suggest that dynamic exploration, of the solution space and complexity, is likely to be limited. The reasoning behind this argument is as follows. Consider a multidimensional solution space which maps to a rugged fitness landscape [137] . In this landscape, competing solution lineages will gradually become extinct through chance processes. So, the solution space explored becomes smaller over time as the population adapts and the diversity of solutions decreases. Ultimately, all solutions may be confined to a small region of the solution space. In a static fitness landscape, this situation is not undesirable because the surviving solution lineages will usually be clustered around an optimum [39]. However, if the fitness landscape is dynamic, the location of optima varies over time, and should lineages become confined to a small area of the solution space, then subsequent selection will locate only optima that are near this area [87]. This is undesirable if new, higher optima arise that are far from pre-existing ones. A related issue is that complex solutions are less likely to be found by chance than simple ones. Complex solutions can be visualised as sharp, isolated peaks on the fitness landscape. Especially for dynamic landscapes, these peaks are most likely to be found when the system explores the solution space widely [87]. Therefore, a self-organising mechanism other than the fitness criteria of users is required to maintain diversity among competing solutions in a Digital Ecosystem.

2.2.5 Stability and Diversity in Complex Adaptive Systems

Ecosystems are often described as Complex Adaptive Systems (CAS), because like them, they are systems made from diverse, locally interacting components that are subject to selection. Other CAS include brains, individuals, economies, and the biosphere. All are characterised by hierarchical organisation, continual adaptation and novelty, and non-equilibrium dynam-

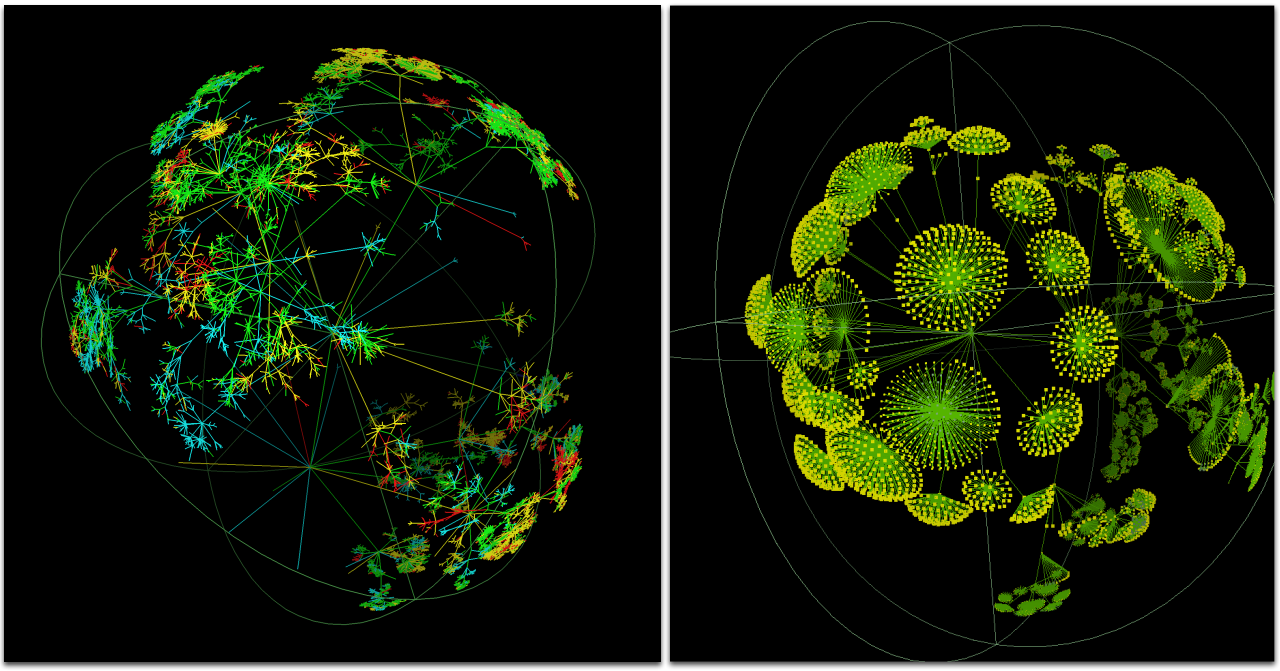

Figure 6: *Ecosystems as Complex Adaptive Systems (modified from [1]): (LEFT) An abstract view of an ecosystem showing the diversity of different populations by the different colours and spacing. (RIGHT) An abstract view of diversity within a population, with the space between points showing genetic diversity and the clustering prevalent.*

ics. These properties lead to behaviour that is non-linear, historically contingent, subject to thresholds, and contains multiple basins of attraction [67].

In the previous subsections, we have advocated Digital Ecosystems that include agent populations evolving by natural selection in distributed environments. Like real ecosystems, digital systems designed in this way fit the definition of CAS. The features of these systems, especially non-linearity and non-equilibrium dynamics, offer both advantages and hazards for adaptive problem-solving. The major hazard is that the dynamics of CAS are intrinsically hard to predict because of the non-linear emergent self-organisation [68]. This observation implies that designing a useful Digital Ecosystem will be partly a matter of trial and error. The occurrence of multiple basins of attraction in CASs suggests that even a system that functions well for a long period may suddenly at some point transition to a less desirable state [35]. For example, in some types of system self-organising mass extinctions might result from interactions among populations, leading to temporary unavailability of diverse solutions [94]. This concern may be addressed by incorporating negative feedback or other mechanisms at the global scale. The challenges in designing an effective Digital Ecosystem are mirrored by the system's potential strengths. Non-linear behaviour provides the opportunity for scalable organisation and the evolution of complex hierarchical solutions, while rapid state transitions potentially allow the system to adapt to sudden environmental changes with minimal loss of functionality [67].

A key question for designers of Digital Ecosystems is how the stability and diversity properties of biological ecosystems map to performance measures in digital systems. For a Digital Ecosystem the ultimate performance measure is user satisfaction, a system-specific property. However, assuming the motivation for engineering a Digital Ecosystem is the development of scalable, adaptive solutions to complex dynamic problems, certain generalisations can be made. Sustained diversity [35], is a key requirement for dynamic adaptation. In Digital Ecosystems, diversity must be balanced against adaptive efficiency because maintaining large numbers of

poorly-adapted solutions is costly. The exact form of this tradeoff will be guided by the specific requirements of the system in question. Stability [67], is likewise, a trade-off: we want the system to respond to environmental change with rapid adaptation, but not to be so responsive that mass extinctions deplete diversity or sudden state changes prevent control.

2.3 Computing of Digital Ecosystems

Based on the understanding of biological ecosystems, from the theoretical biology of the previous subsection, we will now introduce fields from the domain of computer science relevant in the creation of Digital Ecosystems. As we are interested in the digital counterparts for the behaviour and constructs of biological ecosystems, instead of simulating or emulating such behaviour or constructs, we will consider what parallels can be drawn.

The value of creating parallels between biological and computer systems varies substantially depending on the behaviours or constructs being compared, and sometimes cannot be done so convincingly. For example, both have mechanisms to ensure data integrity. In computer systems, that integrity is absolute, data replication which introduces even the most minor change is considered to have failed, and is supported by mechanisms such as the Message-Digest algorithm 5 [109]. Whereas in biological systems, the genetic code is transcribed with a remarkable degree of fidelity; there is, approximately, only one unforced error per one hundred bases copied [79]. There are also elaborate proof-reading and correction systems, which in evolutionary terms are highly conserved [79]. In this example establishing a parallel is unfeasible, despite the relative similarity in function, because the operational control mechanisms in biological and computing systems are radically different, as are the aims and purposes. This is a reminder that considerable finesse is required when determining parallels, or when using existing ones.

We will start by considering Multi-Agent Systems to explore the references to *agents* and *migration*; followed by evolutionary computing and Service-Oriented Architectures for the references to *evolution* and *self-organisation*.

2.3.1 Multi-Agent Systems

A *software agent* is a piece of software that acts, for a user in a relationship of *agency*, autonomously in an environment to meet its designed objectives [136]. A Multi-Agent System (MAS) is a system composed of several *software agents*, collectively capable of reaching goals that are difficult to achieve by an individual agent or monolithic system [136]. Conceptually, there is a strong parallel between the software agents of a MAS and the agents (organisms) of a biological ecosystem, despite the lack of evolution and migration in a MAS. There is an even stronger parallel to a variant of MASs, called *mobile agent systems*, in which agent mobility mirrors the agent migration in biological ecosystems [101].

The term *mobile agent* contains two separate and distinct concepts: mobility and agency [110]. Hence, mobile agents are software agents capable of movement within a network [101]. The mobile agent paradigm proposes to treat a network as multiple agent-*friendly* environments and the agents as programmatic entities that move from location to location, performing tasks for users. So, on each host they visit, mobile agents need software which is responsible for executing the agents, and providing a safe execution environment [101].

Generally, there are three types of design for mobile agent systems [101]: (1) using a specialised operating system, (2) as operating system services or extensions, or (3) as application software. The first approach has the operating system providing the requirements of mobile agent systems directly [124]. The second approach implements the mobile agent system requirements as operating system extensions, taking advantage of existing features of the operating system [54]. Lastly, the third approach builds mobile agent systems as specialised application software that runs on top of an operating system, to provide for the mobile agent functionality,

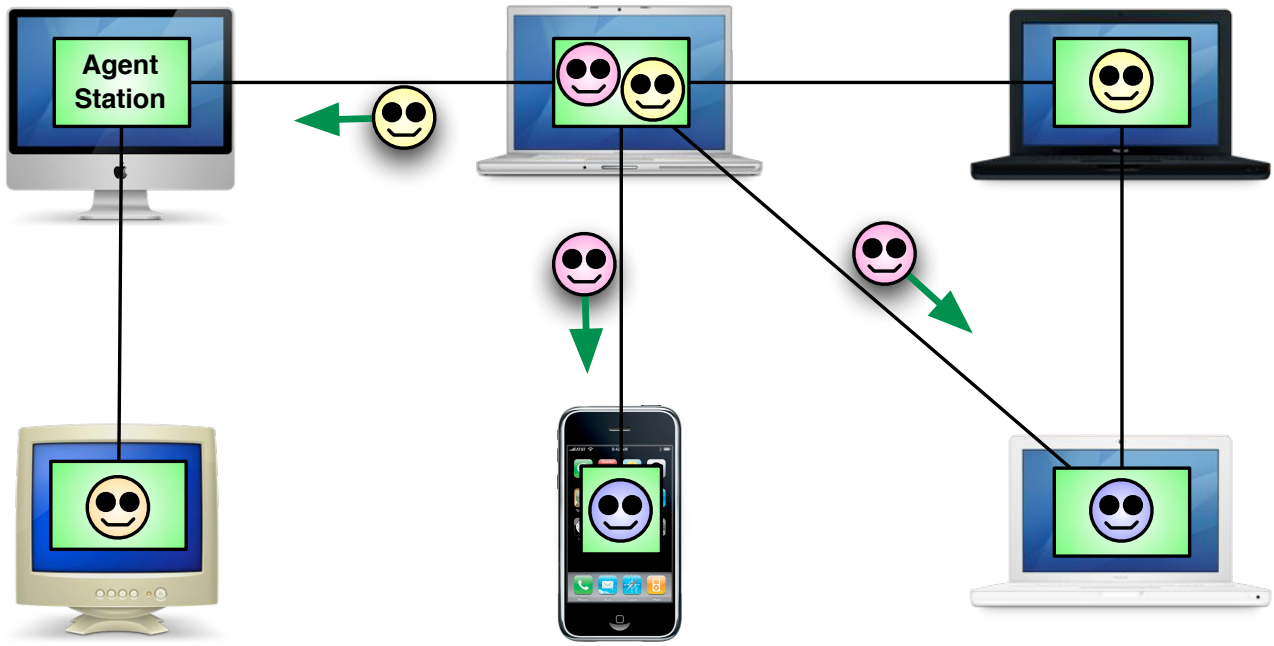

Figure 7: *Mobile Agent System: Visualisation that shows mobile agents as programmes that can migrate from one host to another in a network of heterogeneous computer systems and perform a task specified by its owner. On each host they visit, mobile agents need special software called an agent station, which is responsible for executing the agents and providing a safe execution environment [78].*

with such software being called an *agent station* [78]. In this last approach, each agent station hides the vendor-specific aspects of its host platform, and offers standardised services to visiting agents. Services include access to local resources and applications; for example, web servers or *web services*, the local exchange of information between agents via message passing, basic security services, and the creation of new agents [78]. Also, the third approach is the most platform-agnostic, and is visualised in Figure 7.

2.3.2 Evolutionary Computing

For evolving software in Digital Ecosystems evolutionary computing is the logical field from which to start. In Biologically-Inspired Computing, one of the primary sources of inspiration from nature has been evolution [77]. Evolution has been clearly identified as the source of many diverse and creative solutions to problems in nature [25, 38]. However, it can also be useful as a problem-solving tool in artificial systems. Computer scientists and other theoreticians realised that the selection and mutation mechanisms that appear so effective in biological evolution could be abstracted to be implemented in a computational algorithm [77]. Evolutionary computation is now recognised as a sub-field of artificial intelligence (more particularly computational intelligence) that involves combinatorial optimisation problems [4].

Evolutionary algorithms are based upon several fundamental principles from biological evolution, including reproduction, mutation, recombination (crossover), natural selection, and survival of the fittest. As in biological systems, evolution occurs by the repeated application of the above operators [3]. An evolutionary algorithm operates on a set of individuals, called a population. An *individual*, in the natural world, is an organism with an associated fitness [66]. Candidate solutions to an optimisation problem play the role of individuals in a population, and a cost function determines the environment within which the solutions *live*. The number of

individuals varies between different implementations and may also vary through time during the use of the algorithm. Each individual possesses some characteristics that are defined through its genotype, its genetic composition. These characteristics may be passed on to descendants of that individual [3]. Processes of mutation (small random changes) and crossover (generation of a new genotype by the combination of components from two individuals) may occur, resulting in new individuals with genotypes different from the ancestors they will come to replace. These processes iterate, modifying the characteristics of the population [3]. Which members of the population are kept, or are used as parents for offspring, will often depend upon some external characteristic, called the fitness (cost) function of the population. It is this that enables improvement to occur [3], and corresponds to the fitness of an organism in the natural world [66]. Recombination and mutation create the necessary diversity and thereby facilitate novelty, while selection acts as a force increasing quality. Changed pieces of information resulting from recombination and mutation are randomly chosen. However, selection operators can be either deterministic, or stochastic. In the latter case, individuals with a higher fitness have a higher chance to be selected than individuals with a lower fitness [3].

There are different strands of what has become called evolutionary computation [3]. The first is genetic algorithms. A second strand, evolution strategies, focuses strongly on engineering applications. A third strand, evolutionary programming, originally developed from machine intelligence motivations, and is related to the other two. These areas developed separately for about fifteen years, but from the early nineties they are seen as different representatives (*dialects*) of one technology, called evolutionary computing [32]. In the early nineties, another fourth stream following the general ideas had emerged, called genetic programming [32].

Genetic algorithms [39] implement a population of individuals, each of which possesses a genotype that encodes a candidate solution to a problem. Typically genotypes are encoded as bit-strings, but other encodings have been used in more recent developments of genetic algorithms. Mutation and crossover, along with selection, are then used to choose a solution to a problem. They have proven to be widely applicable, and have resulted in many applications in differing domains [85]. Evolutionary strategies arose out of an attempt by several civil engineers to understand a problem in hydrodynamics [27]. Evolutionary strategies [113] differ from genetic algorithms in operating on real-valued parameters, and historically they have tended not to use crossover as a variational operator, only mutation. However, mutation rates have themselves been allowed to adapt in evolutionary strategies, which is not often the case with genetic algorithms. Evolutionary strategies have also been used for many applications [31].

Evolutionary programming arose distinctly from the first two strands of evolutionary computation, out of an attempt to understand machine intelligence through the evolution of finite state machines [34]. Evolutionary programming [103] emphasises the evolution of the phenotype (instance of a solution) instead of the genotype (genetic material) of individuals, and the relation between the phenotype of parents and offspring, although crossover is not used. Thus, evolutionary programming has some differences in approach from the other major strands of evolutionary computation research. However, there have been many overlaps between the different fields and it too has been applied in many areas [103].

Genetic programming [57] can be considered as a variant of genetic algorithms where individual genotypes are represented by executable programmes. Specifically, solutions are represented as trees of expressions in an appropriate programming language, with the aim of evolving the most effective programme for solving a particular problem. Genetic programming, although the newest form of evolutionary computing, has still proved to be widely applicable [6].

Many important questions remain to be answered in understanding the performance of evolutionary algorithms. For example, current evolutionary algorithms for evolving programmes (genetic programming) suffer from some weaknesses. First, while being moderately successful

at evolving simple programmes, it is very difficult to scale them to evolve high-level software components [75]. Second, the *estimated* fitness of a programme is normally given by a measure of how accurately it computes a given function, as represented by a set of input and output pairs, and therefore there is only a limited guarantee that the evolved programme actually does the intended computation [75]. These issues are particularly important when evolving high-level, complex, structured software.

To evolve high-level software components in Digital Ecosystems, we propose taking advantage of the *native* method of software advancement, human developers, and use evolutionary computing for combinatorial optimisation of the available software services. This involves treating developer-produced software services as the functional building blocks, as the base unit in a genetic-algorithms-based process. Such an approach would require a modular reusable paradigm to software development, such as Service-Oriented Architectures, which are discussed in the following subsection.

2.3.3 Service-Oriented Architectures

Our approach to evolving high-level software applications requires a modular reusable paradigm to software development. Service-oriented architectures (SOAs) are the current state-of-the-art approach, being the current iteration of interface/component-based design from the 1990s, which was itself an iteration of event-oriented design from the 1980s, and before then modular programming from the 1970s [16, 59]. Service-oriented computing promotes assembling application components into a loosely coupled network of services, to create flexible, dynamic business processes and agile applications that span organisations and computing platforms [99]. This is achieved through a SOA, an architectural style that guides all aspects of creating and using business processes throughout their life-cycle, packaged as services. This includes defining and provisioning the infrastructure that allows different applications to exchange data and participate in business processes, loosely coupled from the operating systems and programming languages underlying the applications [93]. Hence, a SOA represents a model in which functionality is decomposed into distinct units (services), which can be distributed over a network, and can be combined and reused to create business applications [99].

A SOA depends upon service-orientation as its fundamental design principle. In a SOA environment, independent services can be accessed without knowledge of their underlying platform implementation [93]. Services reflect a *service-oriented* approach to programming that is based on composing applications by discovering and invoking network-available services to accomplish some task. This approach is independent of specific programming languages or operating systems, because the services communicate with each other by passing data from one service to another, or by co-ordinating an activity between two or more services [99]. So, the concepts of SOAs are often seen as built upon, and the development of, the concepts of modular programming and distributed computing [59].

SOAs allow for an information system architecture that enables the creation of applications that are built by combining loosely coupled and interoperable services [93]. They typically implement functionality most people would recognise as a service, such as filling out an online application for an account, or viewing an online bank statement [59]. Services are intrinsically un-associated units of functionality, without calls to each other embedded in them. Instead of services embedding calls to each other in their source code, protocols are defined which describe how services can talk to each other, in a process known as orchestration, to meet new or existing business system requirements [116]. This is allowing an increasing number of third-party software companies to offer software services, such that SOA systems will come to consist of such third-party services combined with others created in-house, which has the potential to spread costs over many users and uses, and promote standardisation both in and across

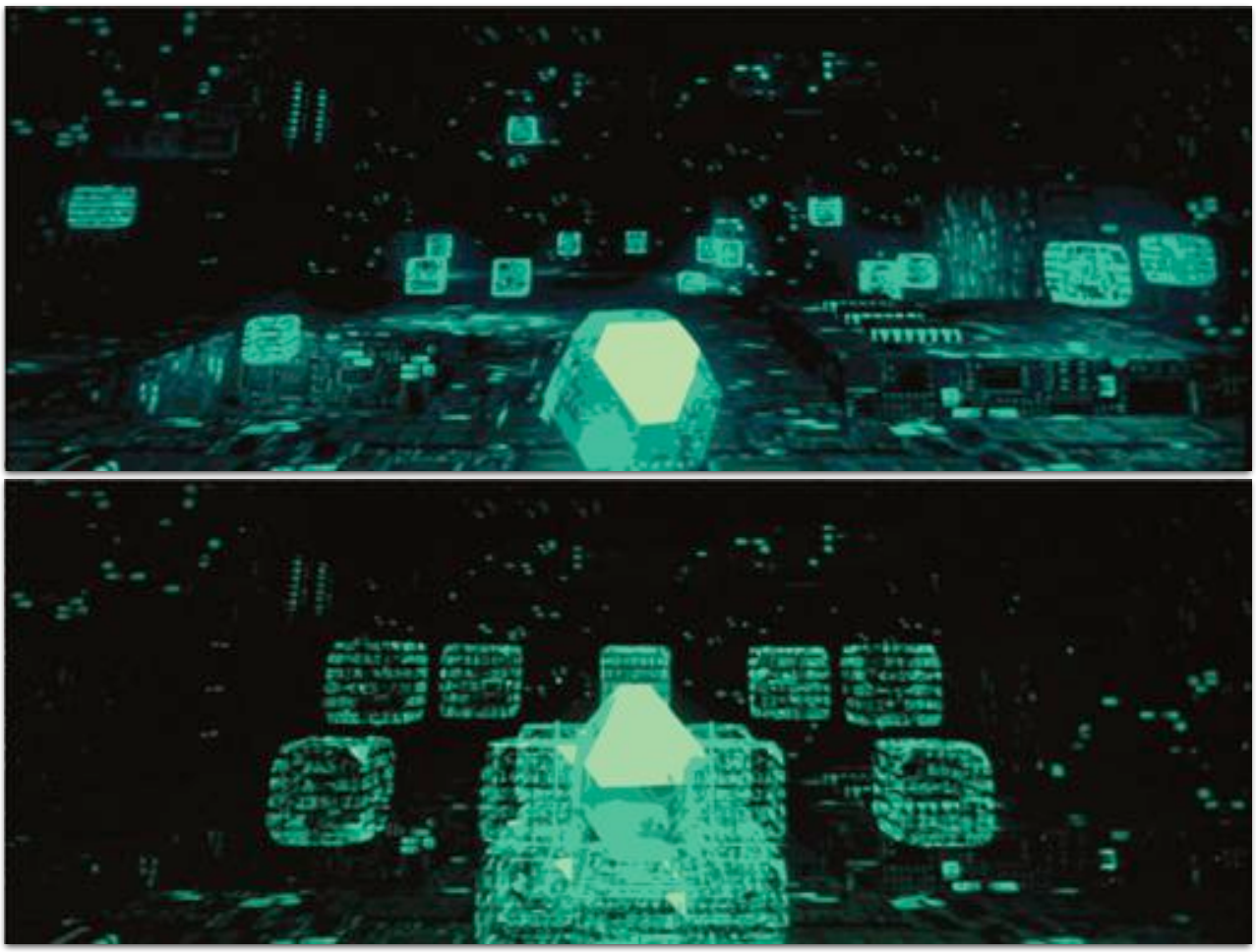

Figure 8: *Service-Oriented Architectures: Abstract visualisations, with the first image showing the loosely joined services as cuboids, and the service orchestration as a polyhedron; and the second image showing their high interoperability and re-usability in forming applications, from the use of standardised interfaces and external service orchestration.*

industries [19]. For example, the travel industry now has a well-defined and documented, set of both services and data, sufficient to allow any competent software engineer to create travel agency software using entirely off-the-shelf software services [56, 18]. Other industries, such as the finance industry, are also making significant progress in this direction [140].

The vision of SOAs assembling application components from a loosely coupled network of services that can create dynamic business processes and agile applications that span organisations and computing platforms, is visualised in Figure 8. It will be made possible by creating compound solutions that use internal organisational software assets, including enterprise information and legacy systems, and combining these solutions with external components residing in remote networks [98]. The great promise of SOAs is that the *marginal cost* of creating the n-th application is virtually zero, as all the software required already exists to satisfy the requirements of other applications. Only their *combination* and *orchestration* are required to produce a new application [125, 86]. The *key* is that the interactions between the *chunks*, are not specified within the *chunks* themselves. Instead, the interaction of services (all of whom are hosted by un-associated peers) is specified by users in an ad-hoc way, with the intent driven by newly emergent business requirements [70].

The pinnacle of SOA interoperability, is the exposing of services on the internet as *web services* [93]. A web service is a specific type of service that is identified by a Uniform Resource

Identifier (URI), whose service description and transport utilise open Internet standards. Interactions between web services typically occur as Simple Object Access Protocol (SOAP) calls carrying eXtensible Markup Language (XML) data content. Interface descriptions of the web services are expressed using the Web Services Definition Language (WSDL) [97]. The Universal Description Discovery and Integration (UDDI) standard defines a protocol for directory services that contain web service descriptions. UDDI enables web service clients to locate candidate services and discover their details. Service clients and service providers utilise these standards to perform the basic operations of SOAs [97]. Service aggregators can then use the Business Process Execution Language (BPEL) to create new web services by defining corresponding compositions of the interfaces and internal processes of existing services [97].

SOA services inter-operate based on a formal definition (or contract, e.g. WSDL) that is independent of the underlying platform and programming language. Service descriptions are used to advertise the service capabilities, interface, behaviour, and quality [97]. The publication of such information about available services provides the necessary means for discovery, selection, binding, and composition of services [97]. The (expected) behaviour of a service during its execution is described by its behavioural description (for example, as a workflow process). Also, included is a quality of service (QoS) description, which publishes important functional and non-functional service quality attributes, such as service metering and cost, performance metrics (response time, for instance), security attributes, integrity (transactional), reliability, scalability, and availability [97]. Service clients (end-user organisations that use some service) and service aggregators (organisations that consolidate multiple services into a new, single service offering) utilise *service descriptions* to achieve their objectives [97]. One of the most important and continuing developments in SOAs is the use of *semantic descriptions* for service discovery, so that a client can discover the services semantically, and then apply transformations to adapt the interface of the services to the interface expected, using already available client software [104].

There are multiple standards available and still being developed for SOAs [130], most notably of recent being REpresentational State Transfer (REST) [116]. The software industry now widely implements a thin SOAP/WSDL/UDDI veneer atop existing applications or components that implement the web services paradigm [98], but the choice of technologies could change with time. Therefore, SOAs and its services are best defined generically, because SOAs are technology agnostic and need not be tied to a specific technology [99]. Within the current and future scope of SOAs, there is clearly potential to *evolve* complex high-level software applications from the modular services of SOAs, instead of the instruction level evolution currently prevalent in genetic programming [58].

2.3.4 Distributed Evolutionary Computing

Having previously introduced evolutionary computing, and the possibility of it occurring within a distributed environment, not unlike those found in mobile agent systems, leads us to consider a specialised form known as distributed evolutionary computing (DEC). The fact that evolutionary computing manipulates a population of independent solutions actually makes it well suited for parallel computation architectures [17]. The motivation for using parallel or distributed evolutionary algorithms is twofold. First, improving the speed of evolutionary processes by conducting concurrent evaluations of individuals in a population. Second, improving the problem-solving process by overcoming difficulties that face traditional evolutionary algorithms, such as maintaining diversity to avoid premature convergence [88, 122]. There are several variants of distributed evolutionary computing, leading some to propose a taxonomy for their classification [95], with there being two main forms [17, 122]:

- multiple-population/coarse-grained migration/island models

- single-population/fine-grained diffusion/neighbourhood models

In the coarse-grained *island* models [71, 17], evolution occurs in multiple parallel sub-populations (islands), each running a local evolutionary algorithm, evolving independently with occasional *migrations* of highly fit individuals among sub-populations. The core parameters for the evolutionary algorithm of the island-models are as follows [71]:

- number of the sub-populations: 2, 3, 4, more
- sub-population homogeneity
  - size, crossover rate, mutation rate, migration interval
- topology of connectivity: ring, star, fully-connected, random
- static or dynamic connectivity
- migration mechanisms:
  - isloated/synchronous/asynchronous
  - how often migrations occur
  - which individuals migrate

Fine-grained *diffusion* models [74, 122] assign one individual per processor. A local neighbourhood topology is assumed, and individuals are allowed to mate only within their neighbourhood, called a *deme*. The demes overlap by an amount that depends on their shape and size, and in this way create an implicit migration mechanism. Each processor runs an identical evolutionary algorithm which selects parents from the local neighbourhood, produces an offspring, and decides whether to replace the current individual with an offspring. However, even with the advent of multi-processor computers, and more recently multi-core processors, which provide the ability to execute multiple threads simultaneously [76], this approach would still prove impractical in supporting the number of agents necessary to create a Digital Ecosystem. Therefore, we shall further consider the *island* models.

An example island-model [71, 17] is visualised in Figure 9, in which there are different probabilities of going from island ① to island ②, as there is of going from island ② to island ①. This allows maximum flexibility for the migration process, and mirrors the naturally inspired quality that although two populations have the same physical separation, it may be easier to migrate in one direction than the other, i.e. fish migration is easier downstream than upstream. The migration of the *island* models is like the notion of migration in nature, being similar to the metapopulation models of theoretical ecology [69]. This model has also been used successfully in the determination of investment strategies in the commercial sector, in a product known as the Galapagos toolkit [131, 22]. However, all the *islands* in this approach work on exactly the same problem, which makes it less analogous to biological ecosystems in which different locations can be environmentally different [9]. We will take advantage of this property later when defining the Ecosystem-Oriented Architecture of Digital Ecosystems.

3 Ecosystem-Oriented Architectures

We will now define the architectural principles of Digital Ecosystems. We will use our understanding of theoretical biology from section 2.2, mimicking the processes and structures of life, evolution, and ecology of biological ecosystems. We will achieve this by combining elements from mobile agents systems, distributed evolutionary computing, and Service-Oriented

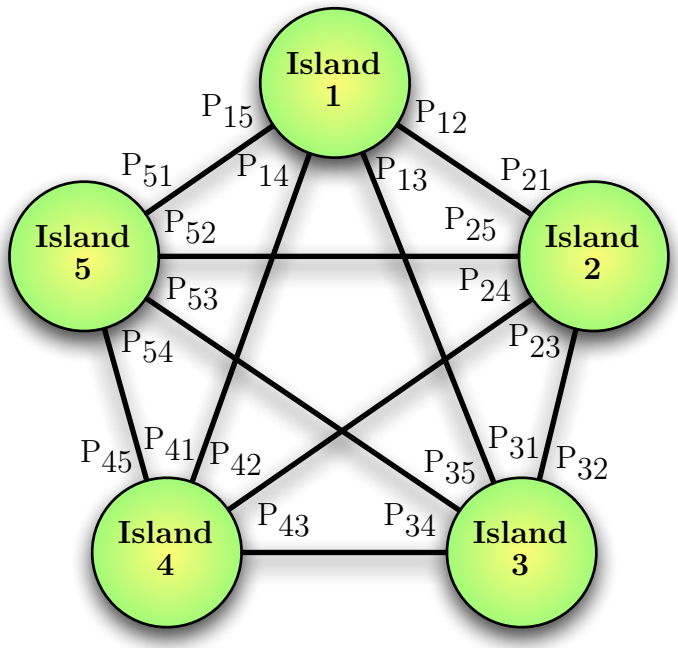

Figure 9: *Island-Model of Distributed Evolutionary Computing [71, 17]: There are different probabilities of going from island ① to island ②, as there is of going from island ② to island ①. This mirrors the naturally inspired quality that although two populations have the same physical separation, it may be easier to migrate in one direction than the other, i.e. fish migration is easier downstream than upstream.*

Architectures from section 2.3, to create a hybrid architecture which is the *digital counterpart of biological ecosxystems*.

We will refer to the agents of Digital Ecosystems as *Agents*, populations as *Populations*, and the habitats as *Habitats*, to distinguish their new hybrid definitions from their original biological and computing definitions.

3.1 Agents

The Agents of the Digital Ecosystem are functionally parallel to the organisms of biological ecosystems, including the behaviour of migration and the ability to be evolved [9], and will be achieved through using a hybrid of different technologies. The ability to migrate is provided by using the paradigm of *agent mobility* from mobile agent systems [101], with the Habitats of the Digital Ecosystem provided by the facilities of *agent stations* from mobile agent systems [78], i.e. a distributed network of locations to migrate to and from. The Habitats, and the Habitat network will be discussed later. The ability of the Agents to be evolved is in two parts: first, by using the interoperability of services from Service-Oriented Architectures [93] to aggregate Agents; and second, the use of evolutionary computing for combinatorial optimisation [32] at the Habitats to evolve optimal aggregations of Agents. The Agents will take advantage of the interoperability of Service-Oriented Architectures [93], by *acting in a relationship of agency* [136] to the user supplied software services, which will be Service-Oriented Architecture compliant [97]. We can then evolve high-level software applications by using evolutionary computing for combinatorial optimisation [3] of the available Agents, or rather the services they represent, in a genetic-algorithms-based process. This makes an Agent, of the Digital Ecosystem, a lightweight entity consisting primarily of a pointer to the software service it represents, including the service's *executable component* and *semantic description*. A software service can be a software service only, e.g. for data encryption, or a software service providing a front-end to a real-world service, e.g. selling books, as shown in Figure 10.

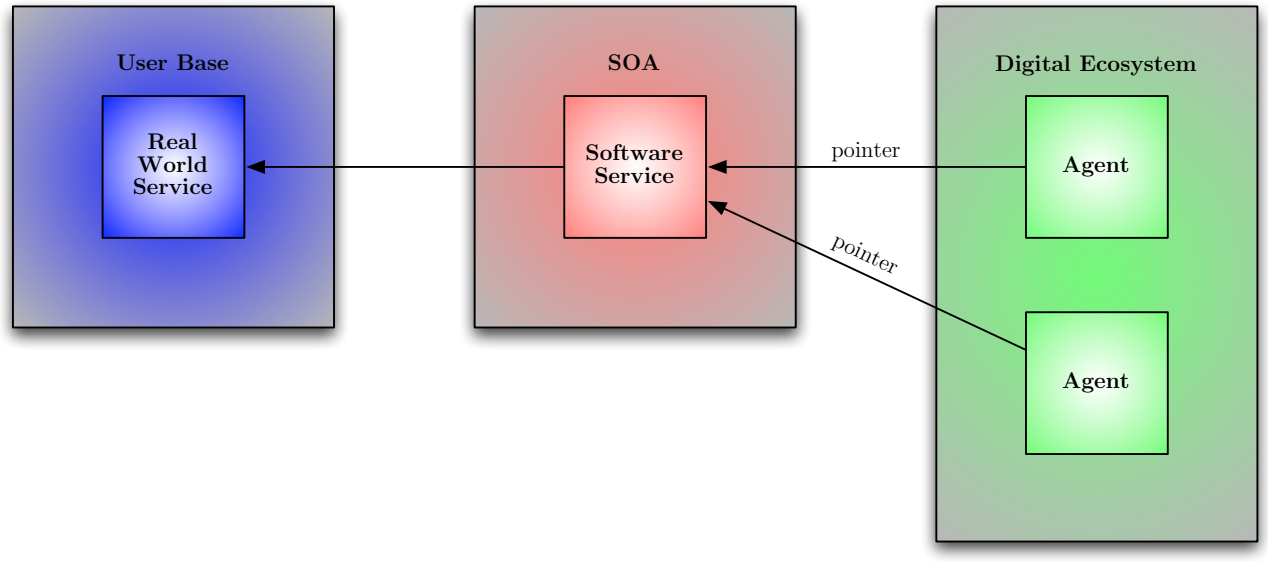

Figure 10: *Agent of the Digital Ecosystem: A lightweight entity consisting primarily of a pointer to the software service it represents, which is itself Service-Oriented Architectures compliant and therefore includes an executable component and semantic description. A software service can be a software service only, e.g. for data encryption, or a software service providing a front-end to a real-world service, e.g. selling books.*

An organism within Digital Ecosystems is an Agent, or an Agent aggregation created using evolutionary optimisation in response to a user request for an application. These Agents will migrate through the Habitat network of the Digital Ecosystem and adapt to find *niches* where they are useful in fulfilling other user requests for applications. The Agents interact, evolve, and adapt over time to the environment, thereby serving the ever-changing requirements imposed by the user base.

The *executable component*, of a service that an Agent represents, is equivalent to the DNA of an organism, whose sequence encodes the genetic information of living organisms and has two primary functions [66]; the holder of virtually all information in inheritance, and the controller of protein synthesis for the construction and operation of its organism. Equivalently, the *executable component* is also the inheritable component from one generation to the next, and defines the objects and behaviour of its service's *run-time instantiation*.

The *genotype* of an individual *describes* the genetic constitution (DNA) of an individual, independent of its physical existence (the phenotype) [66]. Equivalently, the *semantic description*, of a service that an Agent represents, describes the functionality of the *executable component*. The *phenotype* of an individual arises from the combination of an organism's DNA and the environment [66]. Equivalently, the *run-time instantiation*, of a service that an Agent represents, results from instantiating the *executable component* in the *run-time environment*. This differentiation between genotype and phenotype is fundamental for escaping local optima, and is often lacking in artificial evolutionary systems [114], having instead a *one-to-one genotype-phenotype mapping*, in which the phenotype is directly encoded in the genotype with no differentiation provided by instantiation (development) [114]. *Neutral genotype-phenotype mappings* have this differentiation between the genotype and phenotype [115], which more strongly parallels biological evolution [5]. We therefore expect the use of a *neutral genotype-phenotype mapping* to help Digital Ecosystems demonstrate behaviour more akin to biological ecosystems.

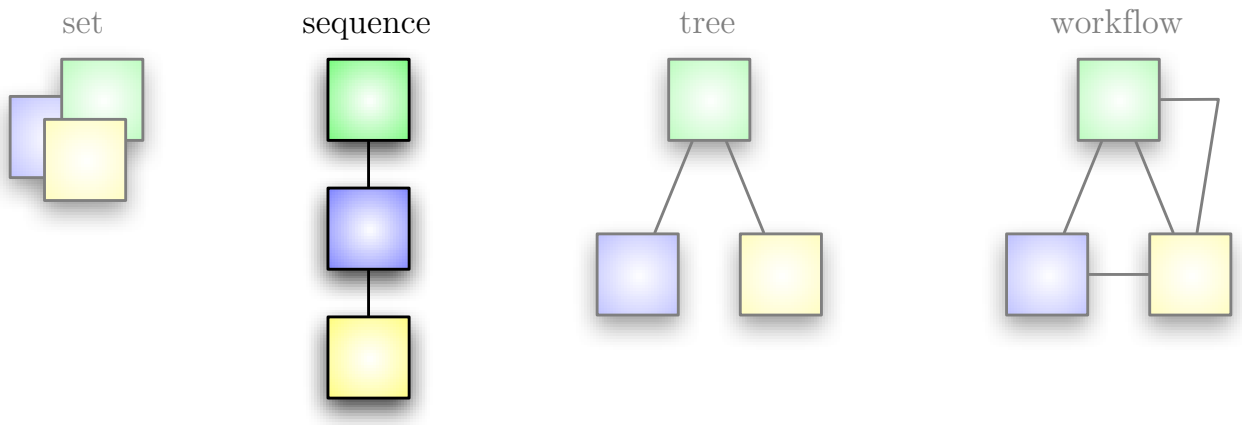

Figure 11: *Structure of Aggregated Agents: The executable component of a service that an Agent represents is equivalent to an organism's DNA and is the gene (functional unit) in the evolutionary process [66]. So, the Agents should be aggregated as a sequence, like the sequencing of genes in DNA [66]. Instead of an unordered set, or, based on service orchestration, into a tree, or workflow.*

3.1.1 Agent Aggregation

The *executable component* of a service that an Agent represents is equivalent to an organism's DNA and is the gene (functional unit) in the evolutionary process [66]. So, the Agents should be aggregated as a *sequence*, like the sequencing of genes in DNA [66]. It could be argued that the Agents should be aggregated as an unordered *set*, or, based on service orchestration, into a *tree*, or *workflow*, as shown in Figure 11. However, the aggregated structure of the Agents should not be the *orchestration* structure of the collection of software services that the Agents represent, not only because the service orchestration of the *run-time instantiation* is application domain-specific (e.g. *trees* in supply chain management [62], *workflows* in the travel industry [11]), but because it would also move it undesirably towards a *one-to-one genotype-phenotype mapping* [114].

3.2 Habitats

The Habitats are the nodes of the Digital Ecosystem, and are functionally parallel to the habitats of a biological ecosystem [66]. Their functionality is provided by using the *agent stations* from mobile agent systems [78], to provide a distributed environment in which Agent migration can occur; with evolutionary computing for Agent interaction, instead of traditional agent interaction mechanisms [136]; and the island-model of distributed evolutionary computing [71] for the connectivity between Habitats. There will be a Habitat for each user, which the users will typically run locally, and through which they will submit requests for applications. Supporting this functionality, Habitats have the following core functions:

- Provide a subset of the Agents and Agent-sequences available globally, relevant to the user that the Habitat represents, and stored in what we will call an Agent-pool (for reasons that will be explained later).
- Accelerate, via the Agent-pool, the Populations instantiated to evolve optimal Agent-sequences in response to user requests for applications.
- Manage the inter-Habitat connections for Agent migration.

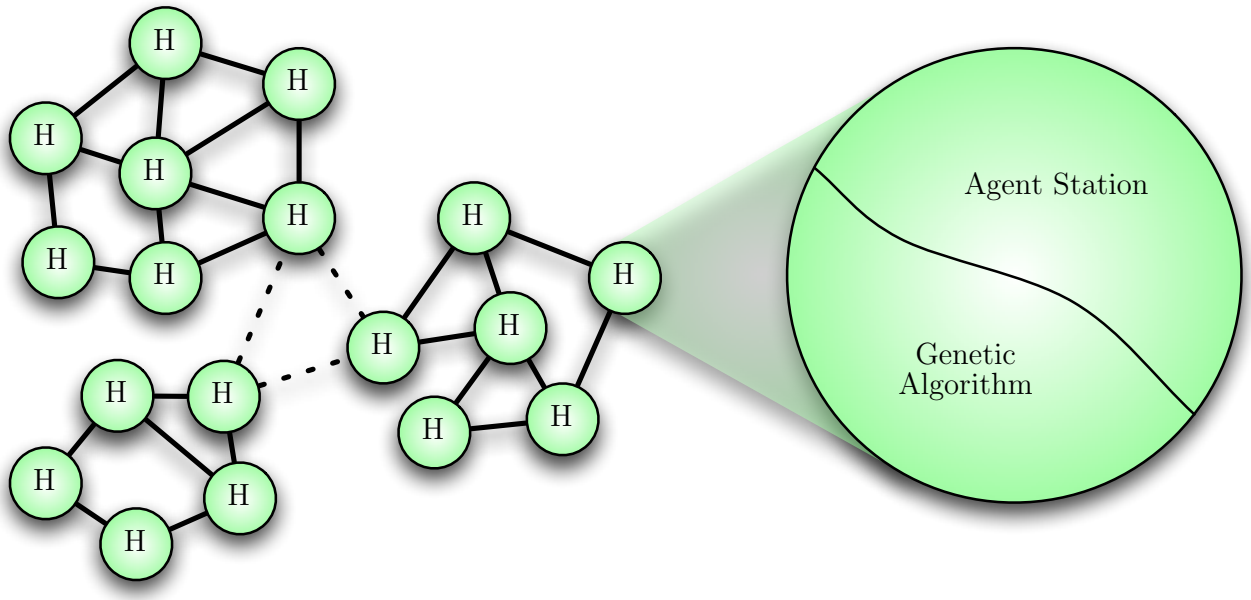

Figure 12: *Habitat Network: Uses the agent stations from mobile agent systems [78], to provide a distributed environment in which Agent migration can occur; with evolutionary computing for Agent interaction, instead of traditional agent interaction mechanisms [136]; and the island-model of distributed evolutionary computing [71] for the connectivity between Habitats.*

- For service providers; manage the distribution of Agents (which represent their services) to other users of the Digital Ecosystem, via the network of interconnected Habitats.

The collection of Agents at each Habitat (peer) will change over time, as the more successful Agents spread throughout the Digital Ecosystem, and as the less successful Agents are deleted. Successive user requests over time to their dedicated Habitats makes this process possible, because the continuous and varying user requests for applications provide a dynamic evolutionary pressure on the Agents, which have to evolve to better satisfy those requests. So, the Agents will recombine and evolve over time, constantly seeking to increase their effectiveness for the user base. The Agent is the base unit of the evolutionary process in Digital Ecosystems, in the same way that the gene is the base unit for evolution in biological ecosystems [9]. So, the collection of Agents at each Habitat provides an Agent-pool, similar to a gene-pool, which is all the genes in a population [66]. Additionally, it also stores Agent-sequences evolved from the Habitat's Populations, and Agent-sequences that migrate to the Habitat from other users' Habitats, because they can potentially accelerate future Populations instantiated to respond to user requests.

The landscape, in energy-centric biological ecosystems, defines the connectivity between habitats [9]. Connectivity of nodes in the digital world is generally not defined by geography or spatial proximity, but by information or semantic proximity. For example, connectivity in a peer-to-peer network is based primarily on bandwidth and information content, and not geography. The island-models of distributed evolutionary computing use an information-centric model for the connectivity of nodes (*islands*) [71]. However, because it is generally defined for one-time use (to evolve a solution to one problem and then stop) it usually has a fixed connectivity between the nodes, and therefore a fixed topology [17]. So, supporting evolution in the Digital Ecosystem, with a multi-objective *selection pressure* (fitness landscape [137] with many peaks), requires a re-configurable network topology, such that Habitat connectivity can be dynamically adapted based on the observed migration paths of the Agents between the users within the Habitat network. Based on the island-models of distributed evolutionary computing [71], each connection between the Habitats is bi-directional and there is a probability associated

with moving in either direction across the connection, with the connection probabilities affecting the rate of migration of the Agents. However, additionally, the connection probabilities will be updated by the success or failure of Agent migration using the concept of Hebbian learning [49]: the Habitats which do not successfully exchange Agents will become less strongly connected, and the Habitats which do successfully exchange Agents will achieve stronger connections. This leads to a topology that adapts over time, resulting in a network that supports and resembles the connectivity of the user base. When we later consider an example user base, we will further discuss a resulting topology.

When a new user joins the Digital Ecosystem, a Habitat needs to be created for them, and most importantly connected to the correct cluster(s) in the Habitat network. A new user's Habitat can be connected randomly to the Habitat network, as it will dynamically reconnect based upon the user's behaviour. User profiling can also be used to help connect a new user's Habitat to the optimal part of the network, by finding a similar user or asking the user to identify a similar user, and then cloning this similar Habitat's connections. When a new Habitat is created, its Agent-Pool should be created by merging the Agent-pools of the Habitats to which it is initially connected.

3.2.1 Agent Migration

The Agents migrate through the interconnected Habitats combining with one another in Populations to meet user requests for applications. The migration path from the current Habitat is dependent on the migration probabilities between the Habitats. The migration of an Agent within the Digital Ecosystem is initially triggered by deployment to its user's Habitat, for distribution to other users who will potentially make use of the service the Agent represents. When a user deploys a service, its representative Agent must be generated and deployed to their Habitat. It is then copied to the Agent-pool of the user's Habitat, and from there the migration of the Agent occurs, which involves migrating (copying) the agent probabilistically to all the connected Habitats. The Agent is copied rather than moved, because the Agent may also be of use to the providing user. The copying of an Agent to a connected Habitat depends on the associated migration probability. If the probability were one, then it would definitely be sent. When migration occurs, depending on the probabilities associated with the Habitat connections, an exact copy of the Agent is made at a connected Habitat. The copy of the Agent is identical until the new Agent's *migration history* is updated, which differentiates it from the original. The successful use of the migrated Agent, in response to user requests for applications, will lead to further migration (distribution) and therefore availability of the Agent to other users.

The connections joining the Habitats are reinforced by successful Agent and Agent-sequence migration. The success of the migration, the *migration feedback*, leads to the reinforcing and creation of migration links between the Habitats, just as the failure of migration leads to the weakening and negating of migration links between the Habitats. The success of migration is determined by the usage of Agents at the Habitats to which they migrate. When an Agent-sequence is found and used in responding to a user request, then the individual Agent *migration histories* can be used to determine where they have come from and update the appropriate connection probabilities. If the Agent-sequence was fully or partly evolved elsewhere, then where the sequence or sub-sequences were created needs to be passed on to the connection probabilities, because the value in an Agent-sequence is the unique ordering and combination it provides of the individual Agents contained within. So, it is necessary to manage the feedback to the connection probabilities for migrating Agent-sequences, and not just the individual Agents contained within the sequence, including the partial use of an Agent-sequence in a newly evolved one. Specifically, the mechanism for *migration feedback* needs to know the Habitats

where migrating Agent-sequences were created, to create new connections or reinforce existing connections to these Habitats. The global effect of the Agent migration and *migration feedback* on the Habitat network is the clustering of Habitats around the communities present within the user base, and will be discussed later in more detail.

The *escape range* is the number of escape migrations available to an Agent upon the risk of death (deletion). If an Agent migrates to a Habitat and is not used after several user requests, then it will have the opportunity to migrate (move not copy) randomly to another connected Habitat. After this happens several times the Agent will be deleted (die). The escape range will be dynamically responsive to the size of the Habitat cluster that the Agent exists within. This creates a dynamic *time-to-live* [23] for the Agents, in which Agents that are used more will live longer and distribute farther than those that are used less.

3.3 Populations

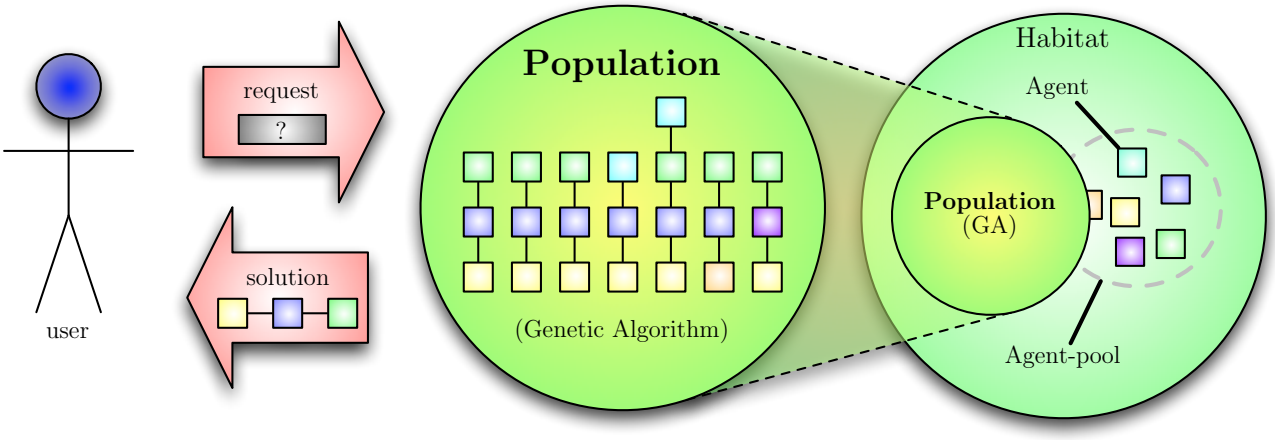

Figure 13: *User Request to the Digital Ecosystem (modified from [61]): A user will formulate queries to the Digital Ecosystem by creating a request as a semantic description, like those being used and developed in Service-Oriented Architectures [104], specifying an application they desire and submitting it to their Habitat. A Population is then instantiated in the user's Habitat in response to the user's request, seeded from the Agents available at their Habitat (the Agent-pool).*

The Populations of the Digital Ecosystem are functionally equivalent to the evolving, self-organising populations of a biological ecosystem, and are achieved through using evolutionary computing. A population in biological ecosystems is all the members of a species that occupy a particular area at a given time [66]. Our Population is also *all the members of a species that occupy a particular area at a given time*, like an island from the island-models of distributed evolutionary computing [71]. The use of distributed evolutionary computing to accelerate the Populations will be explained later.

The users will formulate queries to the Digital Ecosystem by creating a request as a *semantic description*, like those being used and developed in Service-Oriented Architectures [104], specifying an application they desire and submitting it to their Habitat. This description defines a metric for evaluating the *fitness* of a composition of Agents, as a distance function between the *semantic description* of the request and the Agents' *semantic descriptions*. A Population is then instantiated in the user's Habitat in response to the user's request, seeded from the Agents available at their Habitat (the Agent-pool). This allows the evolutionary optimisation to be accelerated in the following three ways: first, the Habitat network provides a subset of the Agents available globally, which is localised to the specific user it represents; second, making use of Agent-sequences previously evolved in response to the user's earlier requests; and third, taking advantage of relevant Agent-sequences evolved elsewhere in response to similar

requests by other users. The Population then proceeds to evolve the optimal Agent-sequence(s) that fulfils the user request, and as the Agents are the base unit for evolution, it searches the available Agent-sequence combination space. For an evolved Agent-sequence that is executed (instantiated) by the user, it then migrates to other peers (Habitats) becoming hosted where it is useful, to combine with other Agents in other Populations to assist in responding to other user requests for applications.

3.3.1 Evolution

Evolution in biological ecosystems leads to both great *diversity* and high *specialisation* of its organisms [9]. In Digital Ecosystems the *diversity* of evolution will provide for the wide range of user needs and allow for quick responses to the changing of these user needs, while the *specialisation* will simultaneously provide solutions which are tailored to fulfil specific user requests. We will consider the issue of *diversity* in a later subsection, because it is achieved through evolution in a distributed environment, which will be discussed later. In biological ecosystems, evolutionary *specialisation* is localised to a population within its micro-habitat, which allows for the creation of niches (high specialisation) [66]. So, a Population is instantiated in the user's own Habitat, where the collection of Agents are chosen for the user, and the micro-Habitat is provided by the user request. There is nothing to preclude more than one Population being instantiated in a user's Habitat at any one time, provided there are computational resources sufficiently available.

A *selection pressure* is the sum aggregate of the forces acting upon a population, resulting in genetic change through natural selection [66]. Those organisms best *fitted* to survive the selection pressures operating upon them will pass on their biological *fitness* to their progeny through the inheritance process [66]. The *fitness* of an individual Agent-sequence within a Population is determined by a *selection pressure*, applied as a *fitness function* [32] instantiated from the user request, and works primarily on comparing the *semantic descriptions* of the Agents with the *semantic description* of the user request. The *pressure* selects for those Agent-sequences that are *fit* and capable of surviving the environment to reproduce, and against those that do not have sufficient *fitness* and therefore die before passing on their *genes*, thereby providing the direction for genetic change. In biology fitness is a measure of an organism's success in its environment [66], and its definition here will be further explained in the next subsection.

Genes are the functional unit in biological evolution [66]; whereas here the functional unit is the Agent. Therefore, the evolutionary process of a Population provides a combinatorial optimisation of the Agents available, when responding to a user request. So, it does not change or mutate the Agents themselves. In biology a *mutation* is a permanent transmissible change (over the generations) in the genetic material (DNA) of an individual, and recombination (e.g. crossover) is the formation within the offspring of alleles (gene combinations), which are not present in the parents [66]. As in genetic algorithms [39], *mutations* will occur by switching Agents in and out of the Agent-sequence structure, and *recombination* (crossover) will occur by performing a crossing of two Agent-sequences.

As the Digital Ecosystem receives more and more sophisticated requests, so more and more complex applications are evolved and become available for use by the users. To achieve this evolution, specifically the Agent-sequence recombination and optimisation, is a very significant challenge, because of the range of services that must be catered for and the potentially huge number of factors that must be considered for creating an applicable *fitness function*. First, to construct ever more complex software solutions, requires modularity, which is provided by the paradigm of service interoperability from Service-Oriented Architectures [93]. Second, two of the most important issues are that of defining *fitness* and managing *bloat*, which we will discuss

next. Finally, there is a huge body of work and continuing research regarding theoretical approaches to evolution ary computing [32], including the extensive use of genetic algorithms for practical real-world problem solving [30]. In defining Digital Ecosystems we should make use of the current state-of-the-art, and future developments, in the areas of evolutionary computing [53] and service interoperability [93].

3.3.2 Fitness

In biology *fitness* is a measure of how successful an organism is in its environment, i.e. its phenotype [66]. The *fitness* of an Agent-sequence within a Population would also, ideally, be based upon its *phenotype*, the *run-time instantiation*, and nothing else. However, such an approach would be impractical, because it is currently unfeasible to execute all the Agent-sequences of a Population at every generation, and not least because of the computational resources that would be required. The other concern is one of practicality, by which we mean that it may not even be possible to perform a live execution for the *executable components* of an Agent-sequence; for example, if they are for buying an item from an online retailer. These are well known issues in evolutionary computing, which is why *fitness functions* are often defined as simulated input/output pairs to test functionality [75]. In Digital Ecosystems we can use historical *usage information*, but this would be insufficient initially, because such information would not be available at the time of an Agent's deployment. However, because each Agent also carries a *semantic description*, a specification of what it does, the *fitness function* can measure a complete Agent-sequence's collective *semantic descriptions* relative to the *semantic description* of a user request. So, initially the *fitness function* should be based primarily on comparing the *semantic descriptions* of the Agent-sequences to the *semantic description* of the user request, ever increasingly augmented with the growing *usage information* available for the Agents. In biological terms the *genotype* will be used as the *phenotype*, combined with any available past fitness of the *phenotype*; with the Agent's *semantic description* (*genotype*) therefore acting as a guarantee of its expected behaviour. So, for any newly deployed Agent a *one-to-one genotype-phenotype mapping* [114] will initially exist, until sufficient *usage information* is available. While the use of such a mapping is undesirable, it is temporary, and necessary to allow Digital Ecosystems to operate effectively.

We have already suggested that the primary driver of the evolutionary process should initially be the extent by which an Agent-sequence can verifiably satisfy the specified requirements. This could be measured probabilistically, or using theorem-proving to validate the system, though automatic theorem proving is notoriously slow [118, 112]. However, there will also be other pressures on the fitness. For example, one may seek the most parsimonious solution to a problem (one that provides exactly the specified features and no more), or the cheapest solution, or one with a good *reputation*. Some aspects of fitness will be implicit in the evolutionary process (e.g. those which are often used will gain more fitness) while others will require explicit measures (e.g. price, or user satisfaction). One way to handle this multiplicity of fitness values (some qualitative) is to explicitly recognise the multi-objective nature of the optimisation problem. In this way, we are seeking not the single best solution, but a range of possible compromises that can be made most optimally. The set of solutions for which there are no better compromises is called the Pareto-set, and evolutionary techniques have been adapted to solve such problems with considerable success [129]. The main point is that selection has to be driven not by an absolute value of fitness, but rather by a notion of what it means for one solution to be better than another. We say one solution dominates another if it is better in at least one respect, and no worse in any of the others [36].

3.3.3 Bloat

If the repetition of Agents is allowed within evolving Agent-sequences, then the search space can become countably infinite, because the nature of the problem to be solved may not allow us to determine what the length of a solution is beforehand. Therefore, a variable length approach must be adopted, which is common in genetic programming [57]. When variable length representations of solutions are used, a well-known phenomenon arises, called *bloat*, in which the individuals of an evolving population tend to grow in size without gaining any additional advantage [65]. The *bloat* phenomenon can cause early termination of an evolutionary process due to the exhaustion of the available memory, and can also significantly reduce performance, because typically longer sequences have higher fitness computation costs [106]. Bloat is not specific to genetic programming, and is inherent in search techniques with discrete variable length representations [63]. It is a fundamental area of research within search-based approaches such as genetic algorithms, genetic programming and other approaches not based on populations such as simulated annealing [63]. However, considerable work on bloat has been done in connection with genetic programming [64, 6], and we believe that the genetic algorithms community generally, and the genetic-algorithms-based approach of our Digital Ecosystems specifically, can benefit directly from this research. While bloat is a phenomenon which was first observed in practice [57], theoretical analyses have been attempted [7]. One should take care with these approaches as implementations will always deal with finite populations, while theoretical approaches often deal with infinite populations [57], and this difference can be important. Yet, both theoretical and empirical approaches are required to understand bloat. There are many factors contributing to bloat, and while the phenomenon may appear simple, the reasons are not. There are several theories to explain why this occurs, and, as we shall discuss, some measures that can be taken for its prevention.

There are several different qualitative theories which attempt to explain bloat, and they can be considered in two groups. First, protection against crossover and bias removal (which can be considered jointly) and second, the nature of programme search spaces [7]. First, near the end of a run a Population consists of mostly fit individuals, and any crossover is likely to be detrimental to the fitness of the offspring. In any sequence of Agents there may be Agents that do not contribute semantically to the complete functionality of the sequence if, for example, their functionality was not requested by the user or if it is duplicated in the sequence; analogously to genetic programming [7], we can call these redundant Agents *bloat*. The genotype can then be grown further without affecting the phenotype if Agents with similar functionality are added; but, as the genotype grows larger, crossover is more likely to transfer redundant Agents to the new off-springs (assuming uniform crossover). Second, above a certain threshold size, the distribution of functionality does not vary with the size of the search space [7]. Thus, if we randomly sample large and small Agent sets, above the size threshold, we are likely to get sequences having the same functionality with the same probability. As a search process progresses, we are therefore more likely to sample bigger sets of Agents, as there are more of them (all other things being equal) and this will give rise to the bloating phenomenon.

Each of the stages of construction of a genetic algorithm (i.e. choice of fitness function, selection method and genetic operator) can affect bloat. It has been shown that even small differences in the fitness function can cause a difference: a single programme glitch in an otherwise flat fitness landscape (of the neutral theory of molecular evolution [55]) is sufficient to drive the average programme size of an infinite population [81]. If a fitness-proportional selection method is used, individuals with zero fitness will be discontinued as they have zero probability of being selected as parents [14]. However, if tournament selection method is used, then there is a finite chance that individuals with zero fitness will be selected to be parents [14]. Finally, the choice of genetic operator affects the size of the programmes which are sampled; *standard crossover on a flat landscape heavily oversamples the shorter programmes* [102]. There are other factors that

may affect bloat, for example, how the population is initialised, or the choice of representation used, such as a neutral genotype-phenotype mapping, which can actually alleviate bloat [83].

Bloat is a fact, whatever the reasons, happening in this type of optimisation and needs to be controlled if the space is to be searched effectively. One solution is to apply a hard limit to the size of the sets that can be sampled [65]: this enables the search algorithm to keep running without having out-of-memory run-time errors, but poses questions on how to set this hard limit. An alternative but similar method is to apply a *parsimony pressure*, where a term is added to the fitness function which chastises big sets in preference for smaller sets [121]. In this approach, individuals larger than the average size are evaluated with a reduced probability, biasing the search to smaller sets, while providing a dynamic limit which adapts to the average size of individuals in a changing population [121].

3.4 The Digital Ecosystem

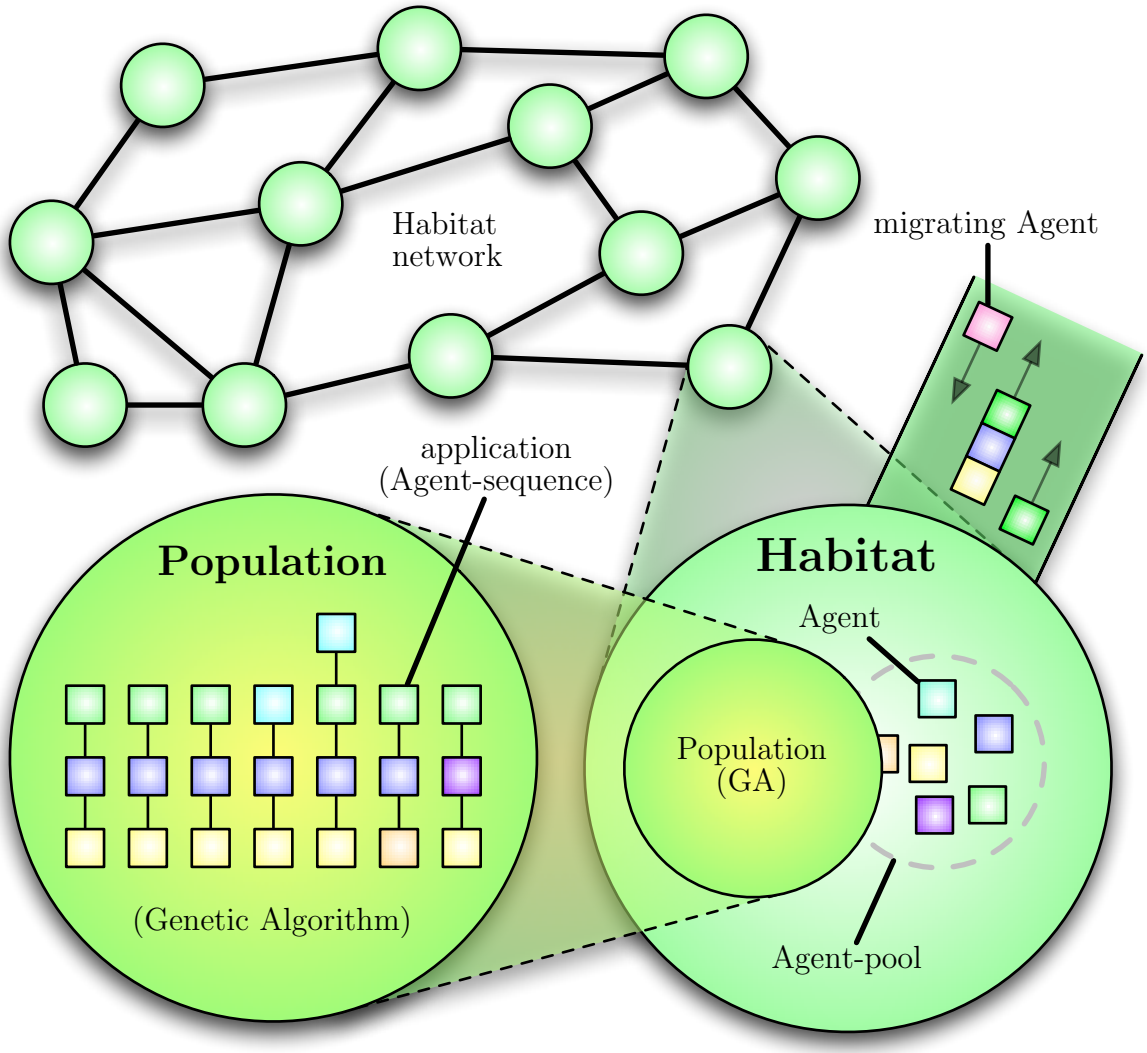

Figure 14: *Digital Ecosystem: A network of interconnected Habitats, combined with the Agents, the Populations, the Agent migration for distributed evolutionary computing, and the environmental selection pressures provided by the user base, then the union of the Habitats creates the Digital Ecosystem. Agents travel along the peer-to-peer connections; in every node (Habitat) local optimisation is performed through an evolutionary algorithm, where the search space is determined by the Agents present at the node.*

The Digital Ecosystem supports the automatic combining of numerous Agents (which represent services), by their interaction in evolving Populations to meet user requests for applications,

in a scalable architecture of distributed interconnected Habitats. The sharing of Agents between Habitats ensures the system is scalable, while maintaining a high evolutionary specialisation for each user. The network of interconnected Habitats is equivalent to the *abiotic* environment of biological ecosystems [9]; combined with the Agents, the Populations, the Agent migration for distributed evolutionary computing, and the environmental selection pressures provided by the user base, then the union of the Habitats creates the Digital Ecosystem, which is summarised in Figure 14. The continuous and varying user requests for applications provide a dynamic evolutionary pressure on the Agent sequences, which have to evolve to better fulfil those user requests, and without which there would be no driving force to the evolutionary self-organisation of the Digital Ecosystem.

In the Digital Ecosystem, local and global optimisations operate concurrently in finding solutions to satisfy different optimisation problems. The global optimisation here is not a decentralised super-peer based control mechanism [108], but the completely distributed peer-to-peer network of the interconnected Habitats, which is not susceptible to the failure of super-peers. This is a novel optimisation technique inspired by biological ecosystems, in which the optimisation works at two levels: a first optimisation, migration of Agents which are distributed in a peer-to-peer network, operating continuously in time; this process feeds a second optimisation, based on evolutionary combinatorial optimisation, operating locally on single peers and is aimed at finding solutions that satisfy locally relevant constraints. The local search is accelerated, through this twofold process, to yield better local optima, as the distributed optimisation already provides a good sampling of the search space, by making use of computations already performed in other peers with similar constraints. This novel form of distributed evolutionary computing will be discussed further below, once we have discussed a topology resulting from an example user base.

If we consider an example user base for the Digital Ecosystem, the use of Service-Oriented Architectures in its definition means that business-to-business (B2B) interaction scenarios [59] lend themselves to being a potential user base for Digital Ecosystems. So, we can consider the Business Ecosystem of Small and Medium sized Enterprise (SME) networks from Digital Business Ecosystems [90], as a specific class of examples for B2B interaction scenarios; and in which the SME users are requesting and providing software services, represented as Agents in the Digital Ecosystem, to fulfil the needs of their business processes. Service-oriented architectures promise to provide potentially huge numbers of services that programmers can combine, via the standardised interfaces, to create increasingly more sophisticated and distributed applications [97]. The Digital Ecosystem extends this concept with the automatic combining of available and applicable services, represented by Agents, in a scalable architecture, to meet user requests for applications. These Agents will recombine and evolve over time, constantly seeking to improve their effectiveness for the user base. From the SME users' point-of-view the Digital Ecosystem provides a network infrastructure where connected enterprises can advertise and search for services (real-world or software only), putting a particular emphasis on the composability of loosely coupled services and their optimisation to local and regional, needs and conditions. To support these SME users the Digital Ecosystem is satisfying the companies' business requirements by finding the most suitable services or combination of services (applications) available in the network. A composition of services is an Agent or Agent-sequence in the Habitat network that can move from one peer (company) to another, being hosted only in those where it is most useful in satisfying the SME users' business needs.

3.4.1 Topology

The Digital Ecosystem allows for the connectivity in the Habitats to adapt to the connectivity within the user base, with a cluster of Habitats representing a community within the user base.

If a user is a member of more than one community, the user's Habitat will be in more than one cluster. This leads to a network topology that will be discovered with time, and which reflects the connectivity within the user base. Similarities in requests by different users will reinforce behavioural patterns, and lead to clustering of the Habitats within the ecosystem, which can occur over geography, language, etc. This will form communities for more effective information sharing, the creation of niches, and will improve the responsiveness of the system. The connections between the Habitats will be self-managed, through the mechanism of Agent migration defined earlier. Essentially, successful Agent migration will reinforce Habitat connections, thereby increasing the probability of future Agent migration along these connections. If a successful multi-hop migration occurs, then a new link between the start and end Habitats can be formed. Unsuccessful migrations will lead to connections (migration probabilities) decreasing, until finally the connection is closed.

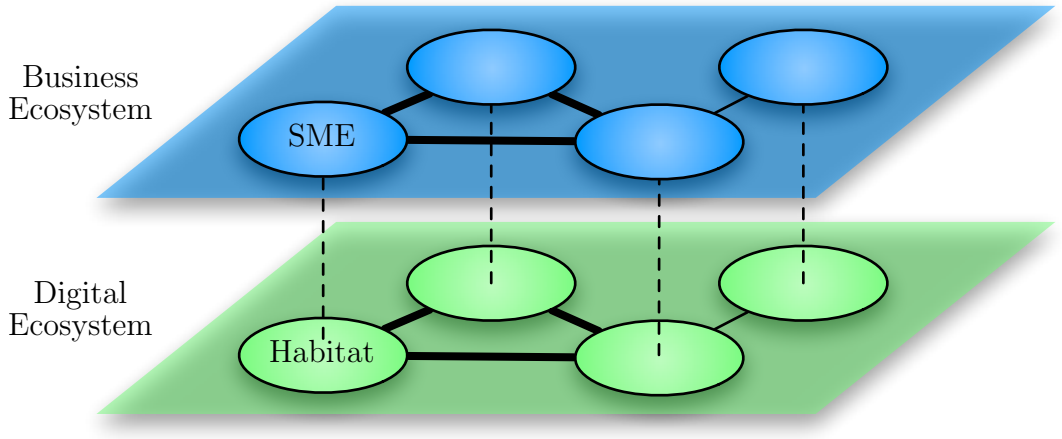

Figure 15: *Digital Business Ecosystem: Business Ecosystem, network of Small and Medium sized Enterprises [90], using the Digital Ecosystem. As the connections between Habitats are reconfigured depending on the connectivity of the user base, the Habitat clustering will therefore be parallel to the business sector communities.*

If we consider the Business Ecosystem, network of Small and Medium sized Enterprises from Digital Business Ecosystems [90], as an example user base; such business networks are typically small-world networks [135, 139]. They have many strongly connected clusters (communities), called *sub-networks* (quasi-complete graphs), with a few connections between these clusters (communities) [133]. Graphs with this topology have a very high clustering coefficient and small characteristic path lengths [133]. As the connections between Habitats are reconfigured depending on the connectivity of the user base, the Habitat clustering will therefore be parallel to the business sector communities, as shown in Figure 15. The communities will cluster over language, nationality, geography, etc. – all depending on the user base.

Fragmentation of the Habitat network can occur, but only if dictated by the structure of the user base. The issue of greater concern is when individual Habitats become totally disconnected, which can only occur under certain conditions. One condition is that the Agents within the Agent-pool consistently fail to satisfy user requests. Another condition being when the Agents and Agent-sequences they share are undesirable to the users that are within the migration range of these Agents and Agent-sequences. Another condition is when the Agents and Agent sequences they share are undesirable to the users that are within the migration range of these Agents and Agent sequences. These scenarios can arise because the Habitat is located within the wrong cluster, in which case the user can be asked to join another cluster within the Habitat network, assuming the user base is of sufficient size to provide a viable alternative.

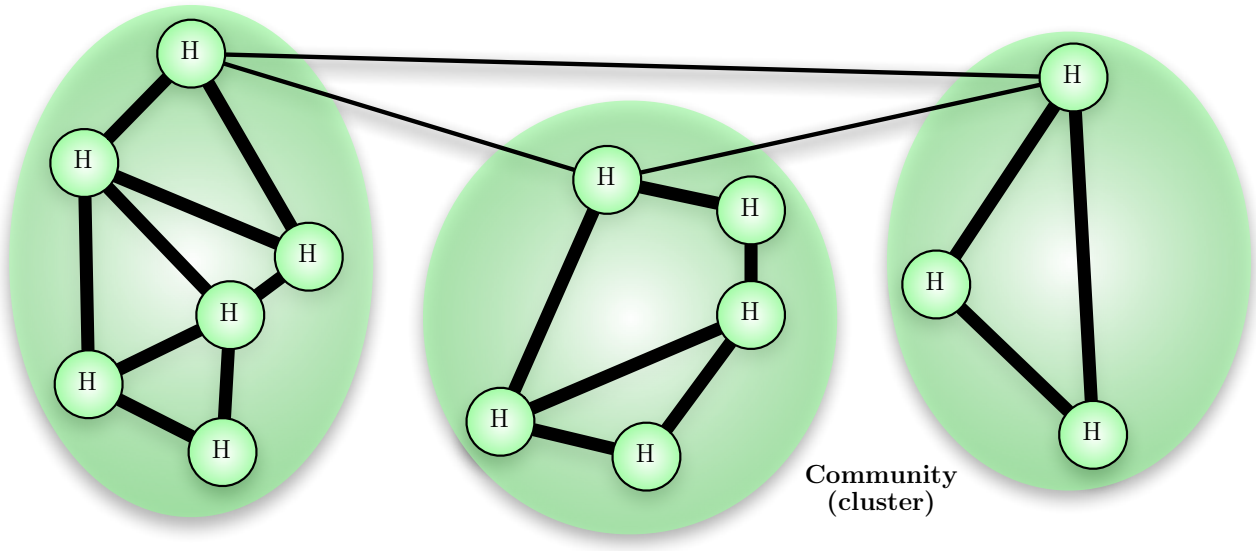

Figure 16: *Habitat Clustering: Topology adapted to the small-world network of a Business Ecosystem of SMEs from Digital Business Ecosystems [90], having many strongly connected clusters (communities), called sub-networks (quasi-complete graphs), with a few connections between these clusters (communities) [133]. Graphs with this topology have a very high clustering coefficient and small characteristic path lengths [133].*

3.4.2 Distributed Evolution

The Digital Ecosystem is a hybrid of Multi-Agent Systems, more specifically of *mobile* agent systems, Service-Oriented Architectures, and *distributed* evolutionary computing, which leads to a novel form of evolutionary computation. The novelty of our approach comes from the evolving Populations being created in response to *similar* requests. So whereas in the island-models of distributed evolutionary computing there are multiple evolving populations in response to one request [71], here there are multiple evolving Populations in response to *similar* requests. In our Digital Ecosystems different requests are evaluated on separate *islands* (Populations), and so adaptation is accelerated by the sharing of solutions between evolving Populations (islands), because they are working to solve similar requests (problems). This is shown in Figure 17, where the dashed yellow lines connecting the evolving Populations indicate similarity in the requests being managed.

If we again consider the Business Ecosystem of Small and Medium sized Enterprises from Digital Business Ecosystems [90] as an example user base, then in Figure 17 the four Habitats, in the left cluster, could be travel agencies, and the three with linked evolving Populations are looking for similar package holidays. So, an optimal solution found and used in one Habitat will be migrated to the other connected Habitats and integrated into any evolving Populations via the local Agent-pools. This will help to optimise the search for similar package holidays at the Habitats of the other travel agencies. This also works in a time-shifted manner, because an optimal solution is stored in the Agent-pool of the Habitats to which it is migrated, being available to optimise a similar request placed later.

The distributed architecture of Digital Ecosystems favours the use of Pareto-sets for fitness determination, because Pareto optimisation for multi-objective problems is usually most effective with spatial distribution of the populations, as partial solutions (solutions to different niches) evolve in different parts of a *distributed population* [28] (i.e. different Populations in different Habitats). Whereas, in a single population, individuals are always interacting with each other, via crossover, which does not allow for this type of specialisation [3].

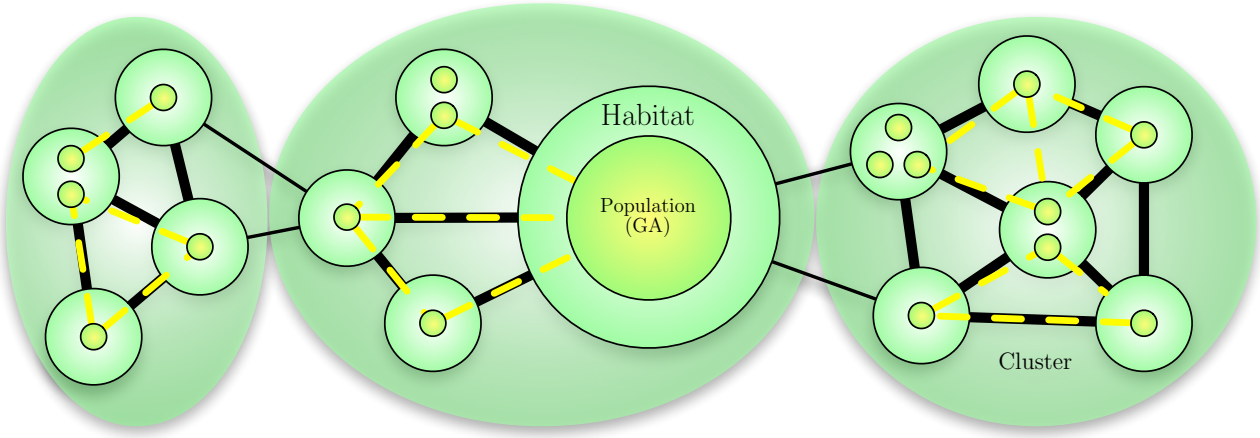

Figure 17: *Distributed Evolution in the Digital Ecosystem: Different requests are evaluated on separate islands (Populations), and so adaptation is accelerated by the sharing of solutions between evolving Populations (islands), because they are working to solve similar requests (problems). The yellow lines connecting the evolving Populations indicate similarity in the requests being managed.*

This approach requires the Digital Ecosystem to have a sufficiently large user base, so that there can be communities within the user base, and therefore allow for similarity in the user requests. Assuming a user base of hundreds of users, then there would be hundreds of Habitats, in which there will be potentially three or more times the number of Populations at any one time. Then there will be thousands of Agents and Agent-sequences (applications) available to meet the requests for applications from the users. In such a scenario, there would be a sufficient number of users for the Digital Ecosystem to find similarity within their requests, and therefore apply our novel form of distributed evolutionary computing.

3.4.3 Agent Life-Cycle

An Agent is created to represent a user's service in the Digital Ecosystem, and its life-cycle begins with deployment to its owner's Habitat for distribution within the Habitat network. The Agent is then migrated to any Habitats connected to the owner's Habitat, to make it available in other Habitats where it could potentially be useful. The Agent is then available to the local evolutionary optimisation, to be used in evolving the optimal Agent-sequence in response to a user request. The optimal Agent-sequence is then registered at the Habitat, being stored in the Habitat's Agent-pool. If an Agent-sequence solution is then executed, an attempt is made to migrate (copy) it to every other connected Habitat, success depending on the probability associated with the connection. The Agent life-cycle is shown in Figure 18.

An Agent can also be deleted if after several successive user requests at a Habitat it remains unused; it will have a small number of *escape migrations*, in which it is not copied, but is randomly moved to another connected Habitat. If the Agent fails to find a *niche* before running out of *escape migrations*, then it will be deleted.

4 Conclusion

We started by reviewing existing *digital ecosystems*, and then introduced biomimicry in computing, Nature Inspired Computing, to create a definition that could be called the *digital counterpart of biological ecosystems*. Then, by comparing and contrasting the relevant theoretical

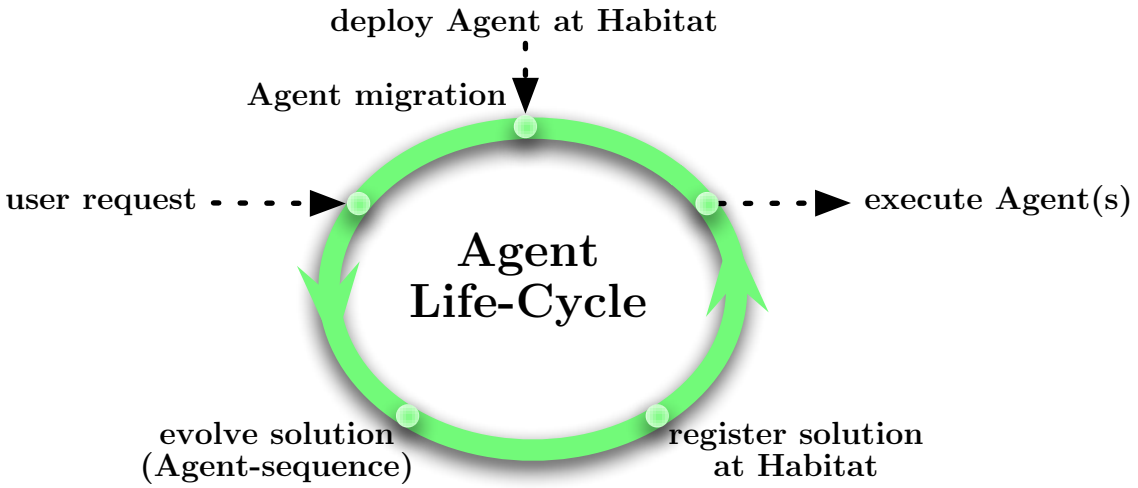

Figure 18: *Agent Life-Cycle: Begins with deployment to its owner's Habitat for distribution within the Habitat network. It can then be used in evolving the optimal Agent-sequence in response to a user request. The optimal Agent-sequence is then registered at the Habitat. If an Agent-sequence solution is then executed, an attempt is made to migrate (copy) it to every other connected Habitat, success depending on the probability associated with the connection.*

ecology, with the anticipated requirements of Digital Ecosystems, we examined how ecological features may emerge in some systems designed for adaptive problem solving. Specifically, we suggested that Digital Ecosystems, like a biological ecosystem, will usually consist of self-replicating agents that interact both with one another and with an external environment [9]. Population dynamics and evolution, spatial and network interactions, and complex dynamic fitness landscapes, will all influence the behaviour of these systems. Many of these properties can be understood via well-known ecological models [73, 51], with a further body of theory that treats ecosystems as Complex Adaptive Systems [67]. These models provide a theoretical basis for the occurrence of self-organisation, in digital and biological ecosystems, resulting from the interactions among the agents and their environment, leading to complex non-linear behaviour [73, 51, 67]; and it is this property that provides the underlying potential for scalable problem-solving in digital environments. Based on the theoretical ecology, we considered fields from the domain of computer science, relevant in the creation of Digital Ecosystems. As we required the digital counterparts for the behaviour and constructs of biological ecosystems, and not their simulation or emulation, we considered parallels using existing and developing technologies to provide their equivalents. This included elements from mobile agent systems [101] to provide a parallel to the *agents* of biological ecosystems and their *migration* to different habitats, and distributed evolutionary computing [71] and Service-Oriented Architectures [93] for the *distribution* and *evolution* of these migrating agents in evolving populations.

Our efforts culminated in the definition of Ecosystem-Oriented Architectures for the creation of Digital Ecosystems, where the Digital Ecosystem supports the automatic combining of numerous Agents (which represent services), by their interaction in evolving Populations to meet user requests for applications, in a scalable architecture of distributed interconnected Habitats. Agents travel along the peer-to-peer connections; in every node (Habitat) local optimisation is performed through an evolutionary algorithm, where the search space is determined by the Agents present at the node. The sharing of Agents between Habitats ensures the system is scalable, while maintaining a high evolutionary specialisation for each user. The network of interconnected Habitats is equivalent to the physical environment of biological ecosystems, and combined with the Agents, the Populations, the Agent migration for distributed evolutionary computing, and the environmental selection pressures provided by the user base; then the union of the Habitats creates the Ecosystem-Oriented Architecture of a Digital Ecosys-

tem. Continuous and varying user requests for applications provide a dynamic evolutionary pressure on the Agent-sequences, which have to evolve to better fulfil those requests, and without which there would be no driving force to the evolutionary self-organisation of the Digital Ecosystem. This represents a novel, cutting-edge approach to *distributed* evolutionary computing, because instead of having multiple populations sharing solutions to find the optimal solution for *one problem*, there are multiple populations to find optimal solutions for *multiple similar problems*. The Business Ecosystem of Small and Medium sized Enterprises from Digital Business Ecosystems [90] was considered as an example user base, because of their adoption of the ecosystems paradigm, and because our use of Service-Oriented Architectures in defining Digital Ecosystems predisposes them to business-to-business interaction scenarios. We have also dealt with critical issues which would otherwise cripple our complex system and prevent it from providing a scalable solution, like bloat in evolutionary processes and points-of-failure in networking topologies. In essence, we are making a system greater than the sum of its parts, expected to show emergent and complex behaviour that cannot be predicted until it is created.

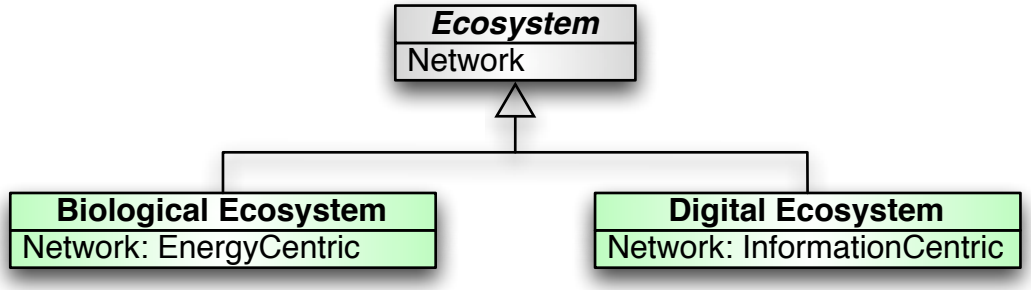

Figure 19: *Hypothetical Abstract Ecosystem Definition: If there were an abstract ecosystem class in the Unified Modelling Language, then the Digital Ecosystem and biological ecosystem classes would both inherit from the abstract ecosystem class, but implement its attributes differently. So, we would argue that the apparent compromises in mimicking biological ecosystems are actually features unique to Digital Ecosystems.*

Creating the digital counterpart of biological ecosystems was not without apparent compromises; the temporary *one-to-one genotype-phenotype mapping* for Agents, and the information-centric dynamically re-configurable network topology. Initially, any newly deployed Agent has a *one-to-one genotype-phenotype mapping* [114], until sufficient *usage (phenotype) information* is amassed for use in *fitness functions*. While the use of such a mapping is undesirable, it is temporary, and necessary to allow the Digital Ecosystem to operate. The Digital Ecosystem requires a re-configurable network topology, to support the constantly changing multi-objective information-centric *selection pressures* of the user base. Hence, using the concept of Hebbian learning [49], Habitat connectivity is dynamically adapted based on the observed migration paths of the Agents within the Habitat network. We would argue that these differences are not compromises, but features unique to Digital Ecosystems. As we discussed earlier, biomimicry, when done well, is not slavish imitation; it is inspiration using the principles which nature has demonstrated to be successful design strategies [13]. Hypothetically, if there were an abstract definition of an ecosystem, defined as an abstract *ecosystem* class, then the *Digital Ecosystem* and *biological ecosystem* classes would both inherit from the abstract *ecosystem* class, but implement its attributes differently, as shown in the Unified Modelling Language class diagram of Figure 19. So, we would argue that the apparent compromises in mimicking biological ecosystems are actually features unique to Digital Ecosystems.

Service-oriented architectures promise to provide potentially huge numbers of services that programmers can combine via standardised interfaces, to create increasingly sophisticated and distributed applications [97]. The Digital Ecosystem extends this concept with the automatic

combining of available and applicable services in a scalable architecture to meet user requests for applications. This is made possible by a fundamental paradigm shift, from a *pull*-oriented approach to a *push*-oriented approach. So, instead of the *pull*-oriented approach of generating applications only upon request in Service-Oriented Architectures [116], the Digital Ecosystem follows a *push*-oriented approach of distributing and composing applications pre-emptively, as well as upon request. Although the use of Service-Oriented Architectures in the definition of Digital Ecosystems provides a predisposition to business [59], it does not preclude other more general uses. The Ecosystem-Oriented Architecture definition of Digital Ecosystems is intended to be inclusive and interoperable with other technologies, in the same way that the definition of Service-Oriented Architectures is with *grid computing and other* technologies [116]. For example, Habitats could be executed using a distributed processing arrangement, such as *cloud computing* [134], which would be possible because the Habitat network topology is information-centric (instead of location-centric).

We have determined the fundamentals for a new class of system, Digital Ecosystems, created through combining understanding from theoretical ecology, evolutionary theory, Multi-Agent Systems, distributed evolutionary computing, and Service-Oriented Architectures. Digital Ecosystems, where the word *ecosystem* is more than just a metaphor, being the digital counterpart of biological ecosystems, and therefore having their desirable properties, such as scalability and self-organisation. It is a complex system that shows emergent behaviour, being more than the sum of its constituent parts.

Acknowledgments

The author would like to thank for their encouragement and suggestions: my supervisor Philippe De Wilde of Heriot-Watt University, my mentor Paolo Dini of the London School of Economics and Political Science, Thomas Heistracher and his group of the Salzburg Technical University, Jonathan Rowe and his group of the University of Birmingham, and Miguel Vidal of Sun Microsystems. The Digital Ecosystem model was constructed through interacting with these people and others. This work has been in part funded by the European Union's 6th Framework Program under EU contract number 507953.

References

[1] A. Agiorgitis. Complex adaptive systems [online]. Available from: `http://www.siliconyogi.com/andreas/it_professional/sol/complexsystems/`.

[2] J. Anderson. *Introduction to Flight.* McGraw-Hill Professional, 2004.

[3] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, 1996.

[4] T. Baeck, D. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation.* CRC Press, 1997.

[5] W. Banzhaf. Genotype-phenotype-mapping and neutral variation–a case study in genetic programming. *Parallel Problem Solving from Nature III*, 866:322–332, 1994.

[6] W. Banzhaf. *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and Its Applications.* Morgan Kaufmann Publishers, 1998.

[7] W. Banzhaf and W. Langdon. Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3:81–91, 2002.

[8] W. Barthlott and C. Neinhuis. Purity of the sacred lotus, or escape from contamination in biological surfaces. *Planta*, 202:1–8, 1997.

[9] M. Begon, J. Harper, and C. Townsend. *Ecology: Individuals, Populations and Communities*. Blackwell Publishing, 1996.

[10] G. Bell. The distribution of abundance in neutral communities. *American Naturalist*, 396:606–617, 2000.

[11] B. Benatallah, M. Dumas, and Q. Sheng. Facilitating the rapid development and scalable orchestration of composite web services. *Distributed and Parallel Databases*, 17:5–37, 2005.

[12] D. Bennett. Digital transformation in the entertainment industry - embracing the fully digital ecosystem. Technical report, Accenture, 2006. Available from: `http://www.accenture.com/NR/rdonlyres/A58111E4-22E5-4DDD-B3DE-FB3741F0052F/0/EmbracingDigitalEco.pdf`.

[13] J. Benyus. *Biomimicry, Innovation Inspired by Nature*. Harper Collins Publishers, 2002.

[14] T. Blickle and L. Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4:361–394, 1996.

[15] S. Bramly. *Leonardo: The Artist and the Man*. Penguin Group, 1994.

[16] E. Canal. Using the ESB with e4: Applying new tendencies to CSC business integration architecture. Technical report, Computer Sciences Corporation, 2007. Available from: `http://www.csc.com/aboutus/leadingedgeforum/knowledgelibrary/uploads/Using%20the%20ESB%20with%20e4%20-%20ERCanal.pdf`.

[17] E. Cantu-Paz. A survey of parallel genetic algorithms. *Réseaux et systèmes répartis, Calculateurs Parallèles*, 10:141–171, 1998.

[18] J. Cardoso and A. Sheth. Introduction to semantic web services and web process composition. In J. Cardoso and A. Sheth, editors, *Semantic Web Services and Web Process Composition*, pages 1–13. Springer, 2004.

[19] A. Chhatpar. Increase business agility through BRM systems and SOA: Best bets to increase ROI on your enterprise applications. Technical report, IBM Devleoper Works, 2008. Available from: `http://download.boulder.ibm.com/ibmdl/pub/software/dw/architecture/ar-brmssoa/ar-brmssoa-pdf.pdf`.

[20] L. Chmbers. *The practical handbook of genetic algorithms: applications*. CRC Press, 2001.

[21] D. Cliff and S. Grand. The creatures global digital ecosystem. *Artificial Life*, 5:77–93, 1999.

[22] Codefarm Software Limited. Codefarm - technology for structured credit [online]. 2008. Available from: `http://www.codefarm.com/` [cited 05 Feb 2008].

[23] D. Comer. *Internetworking with TCP/IP: principles, protocols, and architecture*. Pearson Prentice Hall, 2005.

[24] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: An introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6:86–93, 2002.

[25] C. Darwin. *On the Origin of Species by Means of Natural Selection*. Penguin, Harmondswort, 1969.

[26] L. De Castro. *Fundamentals of Natural Computing: Basic Concepts, Algorithms, And Applications*. CRC Press, 2006.

[27] K. De Jong, D. Fogel, and H. Schwefel. A history of evolutionary computation. In Baeck et al. [4], pages 1–12.

[28] F. de Toro, J. Ortega, J. Fernandez, and A. Diaz. PSFGA: a parallel genetic algorithm for multiobjective optimization. In F. Vajda and N. Podhorszki, editors, *Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 384–391. IEEE Press, 2002.

[29] P. Denning and R. Metcalfe. *Beyond Calculation: The Next Fifty Years of Computing*. Springer, 1997.

[30] E. Ducheyne, B. De Baets, and R. De Wulf. Is fitness inheritance useful for real-world applications. In C. Fonseca, editor, *Evolutionary Multi-criterion Optimization*, pages 31–42. Springer, 2003.

[31] W. Ebeling. Applications of evolutionary strategies. *Systems Analysis Modelling Simulation*, 7:3–16, 1990.

[32] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[33] C. Fiorina. The digital ecosystem [online]. 2000. Available from: `http://www.hp.com/hpinfo/execteam/speeches/fiorina/ceo_worldres_00.html`.

[34] L. Fogel, A. Owens, and M. Walsh. *Artificial Intelligence Through Simulated Evolution*. Wiley, 1966.

[35] C. Folke, S. Carpenter, B. Walker, M. Scheffer, T. Elmqvist, L. Gunderson, and C. Holling. Regime shifts, resilience, and biodiversity in ecosystem management. *Annual Review of Ecology, Evolution, and Systematics*, 35:557–581, 2004.

[36] C. Fonseca and P. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3:1–16, 1995.

[37] N. Forbes. *Imitation of Life: How Biology Is Inspiring Computing*. MIT Press, 2004.

[38] D. Futuyma. *Evolutionary Biology*. Sinauer Associates, 1998.

[39] D. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[40] S. Grand and D. Cliff. Creatures: Entertainment software agents with artificial life. *Autonomous Agents and Multi-Agent Systems*, 1:39–57, 1998.

[41] D. Green and M. Kirley. Adaptation, diversity and spatial patterns. *International Journal of Knowledge-Based Intelligent Engineering Systems*, 4:184–190, 2000.

[42] D. Green, N. Klomp, G. Rimmington, and S. Sadedin. *Complexity in Landscape Ecology*. Springer, 2006.

[43] D. Green, T. Leishman, and S. Sadedin. Dual phase evolution: a mechanism for self-organization in complex systems. In A. Minai, D. Braha, and Y. Bar-Yam, editors, *International Conference on Complex Systems*. Springer, 2006.

[44] D. Green, D. Newth, and M. Kirley. Connectivity and catastrophe - towards a general theory of evolution. In M. Bedau, editor, *International Conference on Artificial Life*, pages 153–161. MIT Press, 2000.

[45] D. Green and S. Sadedin. Interactions matter- complexity in landscapes and ecosystems. *Ecological Complexity*, 2:117–130, 2005.

[46] I. Hanski. *Metapopulation Ecology*. Oxford University Press, 1999.

[47] I. Hanski and O. Ovaskainen. Metapopulation theory for fragmented landscapes. *Theoretical Population Biology*, 64:119–127, 2003.

[48] C. Hastrich. Biomimicry: A tool for innovation [online]. 2007. Available from: `http://www.biomimicryinstitute.org/about-us/biomimicry-a-tool-for-innovation.html`.

[49] D. Hebb. *The Organization of Behavior*. Wiley, 1949.

[50] M. Hollis. *The Philosophy of Social Science: An Introduction*. Cambridge University Press, 1994.

[51] S. Hubbell. *The Unified Neutral Theory of Biodiversity and Biogeography*. Princeton University Press, 2001.

[52] M. Iansiti and R. Levien. *The Keystone Advantage: What the New Dynamics of Business Ecosystems Mean for Strategy, Innovation, and Sustainability*. Harvard Business School Press, 2004.

[53] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 9:3–12, 2005.

[54] D. Johansen, R. van Renesse, and F. Schneider. Operating system support for mobile agents. In *Workshop on Hot Topics in Operating Systems*, pages 42–45. IEEE Press, 1995.

[55] M. Kimura. *The neutral theory of molecular evolution.* Cambridge University Press, 1983.

[56] A. Kotok. *ebXML: The New Global Standard for Doing Business Over the Internet*. Sams Publishing, 2001.

[57] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[58] J. Koza. Overview of genetic programming. In *Genetic Programming: On the Programming of Computers by Means of Natural Selection* [57], pages 73–78.

[59] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall, 2004.

[60] M. Kulkarni and R. Kreutzer. Building your own digital ecosystem: a holistic approach to enterprise integration. Technical report, Syntel, 2006. Available from: `http://www.syntelinc.com/uploadedFiles/Syntel_DigitalEcosystem.pdf`.

[61] T. Kurz and T. Heistracher. Simulation of a self-optimising digital ecosystem. In *Digital EcoSystems and Technologies Conference*, pages 165–170. IEEE Press, 2007.

[62] D. Lambert and M. Cooper. Issues in supply chain management. *Industrial Marketing Management*, 29:65–83, 2000.

[63] W. Langdon. The evolution of size in variable length representations. In P. Simpson, editor, *International Conference on Evolutionary Computation*, pages 633–638. IEEE Press, 1998.

[64] W. Langdon. *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!* Kluwer Publishers, 1998.

[65] W. Langdon and R. Poli. Fitness causes bloat. In P. Chawdhry, R. Roy, and R. Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer, 1997.

[66] E. Lawrence. *Henderson's dictionary of biological terms.* Pearson Education, 2005.

[67] S. Levin. Ecosystems and the biosphere as complex adaptive systems. *Ecosystems*, 1:431–436, 1998.

[68] S. Levin. *Fragile dominion: complexity and the commons.* Perseus Books Group, 1999.

[69] R. Levins. Some demographic and genetic consequences of environmental heterogeneity for biological control. *Bulletin of the Entomological Society of America*, 15:237–240, 1969.

[70] F. Leymann, D. Roller, and M. Schmidt. Web services and business process management. *IBM Systems Journal*, 41:198–211, 2002.

[71] S. Lin, W. Punch III, and E. Goodman. Coarse-grain parallel genetic algorithms: categorization and new approach. In *Symposium on Parallel and Distributed Processing*, pages 28–37. IEEE Press, 1994.

[72] K. Lyytinen and Y. Yoo. The next wave of nomadic computing: A research agenda for information systems research. *Sprouts: Working Papers on Information Systems*, 1:1–20, 2001.

[73] R. MacArthur and E. Wilson. *The Theory of Island Biogeography.* Princeton University Press, 1967.

[74] B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithms. In J. Schaffer, editor, *International Conference on Genetic Algorithms*, pages 428–433. Morgan Kaufmann Publishers, 1989.

[75] T. Mantere and J. Alander. Evolutionary software engineering, a review. *Applied Soft Computing*, 5:315–331, 2005.

[76] J. Markoff. Faster chips are leaving programmers in their dust. Technical report, New York Times, 2007. Available from: `http://www.nytimes.com/2007/12/17/technology/17chip.html`.

[77] P. Marrow. Nature-inspired computing technology and applications. *BT Technology Journal*, 18:13–23, 2000.

[78] F. McCabe and K. Clark. April-agent process interaction language. In M. Wooldridge and N. Jennings, editors, *Intelligent Agents: Workshop on Agent Theories, Architectures, and Languages*, pages 324–340. Springer, 1994.

[79] S. McCulloch, R. Kokoska, O. Chilkova, C. Welch, E. Johansson, P. Burgers, and T. Kunkel. Enzymatic switching for efficient and accurate translesion DNA replication. *Nucleic Acids Research*, 32:4665–4675, 2004.

[80] S. McIlraith, C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16:46–53, 2001.

[81] N. McPhee and R. Poli. A schema theory analysis of the evolution of size in genetic programming with linear representations. In J. Miller, M. Tomassini, P. Lanzi, C. Ryan, A. Tettamanzi, and W. Langdon, editors, *European Conference on Genetic Programming*, pages 108–125. Springer, 2001.

[82] N. Milanovic and M. Malek. Current solutions for web service composition. *IEEE Internet Computing*, 8:51–59, 2004.

[83] J. Miller. What bloat? cartesian genetic programming on boolean problems. In L. Spector, editor, *Genetic and Evolutionary Computation Conference*, pages 295–302. Morgan Kaufmann Publishers, 2001.

[84] S. Miller. Aspect-oriented programming takes aim at software complexity. *Computer*, 34:18–21, 2001.

[85] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[86] G. Modi. Service oriented architecture & web 2.0. Technical report, Guru Tegh Bahadur Institute of Technology, 2007. Available from: `http://www.gsmodi.com/files/SOA_Web2_Report.pdf`.

[87] R. Morrison. *Designing Evolutionary Algorithms For Dynamic Environments*. Springer, 2004.

[88] H. Muhlenbein. Evolution in time and space - the parallel genetic algorithm. *Foundations of Genetic Algorithms*, 1:316–337, 1991.

[89] F. Nachira. Towards a network of digital business ecosystems fostering the local development. Technical report, Directorate General Information Society and Media, European Commission, 2002. Available from: `http://www.digital-ecosystems.org/doc/discussionpaper.pdf`.

[90] F. Nachira, P. Dini, and A. Nicolai. A network of digital business ecosystems for europe: Roots, processes and perspectives. In Nachira et al. [91], pages 1–20.

[91] F. Nachira, A. Nicolai, P. Dini, M. Le Louarn, and L. Rivera León, editors. *Digital Business Ecosystems*. European Commission, 2007.

[92] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of web services. In *international conference on World Wide Web*, pages 77–88. ACM Press, 2002.

[93] E. Newcomer and G. Lomow. *Understanding SOA with web services.* Addison-Wesley, 2005.

[94] M. Newman. A model of mass extinction. *Journal of Theoretical Biology*, 189:235–252, 1997.

[95] M. Nowostawski and R. Poli. Parallel genetic algorithm taxonomy. In L. Jain, editor, *International Conference on Knowledge-Based Intelligent Information Engineering Systems*, pages 88–92. IEEE Press, 1999.

[96] M. Papazoglou. Agent-oriented technology in support of e-business. *Communications of the ACM*, 44:71–77, 2001.

[97] M. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Communications of the ACM*, 46:25–28, 2003.

[98] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40:38–45, 2007.

[99] M. Papazoglou. Service-oriented computing: concepts, characteristics and directions. In T. Catarci, M. Mecella, J. Mylopoulos, and M. Orlowska, editors, *International Conference on Web Information Systems Engineering*, pages 3–12. IEEE Press, 2003.

[100] R. Pennock. Evolving intelligence [online]. 2006. Available from: `http://www.msu.edu/~pennock5/research/EI.html`.

[101] V. Pham and A. Karmouch. Mobile software agents: an overview. *IEEE Communications Magazine*, 36:26–37, 1998.

[102] R. Poli and N. McPhee. Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. In J. Miller, editor, *European conference on Genetic Programming*, pages 126–142. Springer, 2001.

[103] V. Porto. Evolutionary programming. In T. Baeck, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, pages 89–102. CRC Press, 2000.

[104] P. Rajasekaran, J. Miller, K. Verma, and A. Sheth. Enhancing web services description and discovery to facilitate composition. In J. Cardoso and A. Sheth, editors, *Semantic Web Services and Web Process Composition*, pages 55–68. Springer, 2004.

[105] J. Rao and X. Su. A survey of automated web service composition methods. In J. Cardoso and A. Sheth, editors, *Semantic Web Services and Web Process Composition*, pages 43–54. Springer, 2004.

[106] A. Ratle and M. Sebag. Avoiding the bloat with probabilistic grammar-based genetic programming. In P. Collet, C. Fonlupt, J.-K. Hao, E. Lutton, and M. Schoenauer, editors, *International Conference on Artificial Evolution*, pages 255–266. Springer, 2001.

[107] A. Redmore and M. Griffin. *Advanced Level and Advanced Special Level Biology.* Longman Group Limited, 1994.

[108] J. Risson and T. Moors. Survey of research towards robust peer-to-peer networks: Search methods. *Computer Networks*, 50:3485–3521, 2006.

[109] R. Rivest. The MD5 message-digest algorithm. Technical report, MIT, 1992. Available from: `http://people.csail.mit.edu/rivest/Rivest-MD5.txt`.

[110] K. Rothermel and F. Hohl. Mobile agents. In A. Kent and J. Williams, editors, *Encyclopedia for Computer Science and Technology*, volume 40, pages 155–176. CRC Press, 1999.

[111] M. Salmon. *Introduction to the Philosophy of Science*. Hackett Publishing, 1999.

[112] J. Schumann. *Automated Theorem Proving in Software Engineering*. Springer, 2001.

[113] H. Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. Wiley, 1995.

[114] M. Shackleton, R. Shipma, and M. Ebner. An investigation of redundant genotype-phenotype mappings and their role in evolutionary search. In *Congress on Evolutionary Computation*, pages 493–500. IEEE Press, 2000.

[115] R. Shipman, M. Shackleton, and I. Harvey. The use of neutral genotype-phenotype mappings for improved evolutionary search. *BT Technology Journal*, 18:103–111, 2000.

[116] M. Singh and M. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. Wiley, 2005.

[117] A. Sizling and D. Storch. Power-law species-area relationships and self-similar species distributions within finite areas. *Ecology Letters*, 7:60–68, 2004.

[118] J. Slagle. Automated theorem-proving for theories with simplifiers, commutativity, and associativity. *Journal of the ACM*, 21:622–642, 1974.

[119] R. Solé, D. Alonso, and A. McKane. Self-organized instability in complex ecosystems. *Philosophical Transactions: Biological Sciences*, 357:667–681, 2002.

[120] K. Soraku-gun. An evolutionary approach to synthetic biology: Zen and the art of creating life. In C. Langton, editor, *Artificial Life: An Overview*, pages 195–226. MIT Press, 1995.

[121] T. Soule and J. Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation*, 6:293–309, 1998.

[122] J. Stender. *Parallel Genetic Algorithms: Theory and Applications*. IOS Press, 1993.

[123] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 2005.

[124] M. Svahnberg. Background analysis and design of an agent-based operating system. Master's thesis, University of Karlskrona/Ronneby, 1998. Available from: `http://www.ide.hk-r.se/~nesse/ABOS/ABOS_thesis.pdf`.

[125] Q. Tang. *Economics Of Web Service Provisioning: Optimal Market Structure And Intermediary Strategies*. PhD thesis, University of Florida, 2004.

[126] D. Tilman and P. Kareiva. *Spatial Ecology: The Role of Space in Population Dynamics and Interspecific Interactions*. Princeton University Press, 1997.

[127] S. Valverde-Castillo. Software design as an adaptive walk [online]. 2004. Available from: `http://web.archive.org/web/20041206173432/http://complex.upf.es/~sergi/`.

[128] M. Vandenberghe. Digital ecosystem solution - the business factory. Technical report, XeWOW, 2006. Available from: http://themaddesigner.free.fr/XeWOW%20White%20Paper.pdf.

[129] D. Veldhuizen and G. Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation*, 8:125–147, 2000.

[130] B. Violino. How to navigate a sea of SOA standards [online]. 2007. Available from: http://www.cio.com/article/104007/How_to_Navigate_a_Sea_of_SOA_Standards.

[131] M. Ward. Life offers lessons for business [online]. 2004. Available from: http://news.bbc.co.uk/1/hi/technology/3752725.stm.

[132] M. Waterman. *Introduction to Computational Biology: Maps, Sequences and Genomes*. CRC Press, 1995.

[133] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[134] A. Weiss. Computing in the clouds. *netWorker*, 11:16–25, ACM Press, 2007.

[135] D. White and M. Houseman. The navigability of strong ties: Small worlds, tie strength, and network topology. *Complexity*, 8:72–81, 2002.

[136] M. Wooldridge. *Introduction to MultiAgent Systems*. Wiley, 2002.

[137] S. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In D. Jones, editor, *International Congress on Genetics*, pages 356–366. Brooklyn botanic garden, 1932.

[138] Ximbiotix. About the digital ecosystem [online]. 2005. Available from: http://www.ximbiotix.com/desktop/the-digital-ecosystem/about-the-digital-ecosystem.cfm [cited 26 Jan 2008].

[139] X. Yang. Chaos in small-world networks. *Physical Review E*, 63:046206:1–4, 2001.

[140] O. Zimmermann, S. Milinski, M. Craes, and F. Oellermann. Second generation web services-oriented architecture in production in the finance industry. In J. Vlissides and D. Schmidt, editors, *Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 283–289. ACM Press, 2004.